AD-A284 871

TASK: UU03
CDRL: 05156
February 1993

Reuse Library Framework Modeler Tutorial

Informal Technical Data

STARS-UC-05156/020/00
February 1993

TASK: U03
CDRL: 05156
February 1993

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

RLF Modeler Tutorial

STARS-UC-05156/020/00
February 1993

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0011

Prepared for:

Electronic Systems Center
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: U03
CDRL: 05156
February 1993

Data ID: STARS-UC-05156/020/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Developed by: Paramax Systems Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

TASK: U03
CDRL: 05156
February 1993

INFORMAL TECHNICAL REPORT
RLF Modeler Tutorial

Principal Author(s):

Jim Solderitsch *Date*

Approvals:

Task Manager *Richard E. Creps* *Date*

(Signatures on File)

TASK: U03
CDRL: 05156
February 1993

INFORMAL TECHNICAL REPORT
RLF Modeler Tutorial

Change Record:

Data ID	*Description of Change*	*Date*	*Approval*
STARS-UC-05156/020/00	Reissued: Minor changes to accompany RLF v4.1 release	February 1993	*on file*
STARS-UC-05156/010/00	Original Issue	November 1992	*on file*

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Informal Technical Report

4. TITLE AND SUBTITLE

RLF Modeler Tutorial

5. FUNDING NUMBERS

F19628-88-D-0031

6. AUTHOR(S)

Paramax Corporation

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Paramax Corporation
1210 Sunrise Valley Drive
Reston, VA 22090

8. PERFORMING ORGANIZATION REPORT NUMBER

STARS-UC-05156/020/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
Headquarter, Electronic Systems
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

05156

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This package is part of an evolving series of orientation packages that constitute a comprehensive RLF training program. Eventually, this program will include material for three distinct categories of RLF users:

- end users of RLF-based applications, concentrating on those employing the RLF Graphical Browser (RLF GB);
- maintainers and administrators of RLF-based applications and in particular the underlying knowledge bases on which the applications depend for much of their power;
- RLF model developers and application designers.

14. SUBJECT TERMS

15. NUMBER OF PAGES 80

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

1 Introduction

The RLF is a knowledge-based system, developed by Paramax with support by the STARS program, whose primary application has been the design, implementation, and deployment of domain-specific reuse library systems. A reuse library supports software engineers by enabling them to quickly locate software assets (e.g. requirements, designs, code modules, test plans etc.) that can be of use in their construction of a software system.

Slide 1

What is the RLF?

- RLF stands for Reuse Library Framework.
- The RLF is a system, written in Ada, that enables the creation of knowledge-based systems.
- In particular, the RLF has been applied to the creation of domain-specific reuse libraries.

A domain is an application area, typically the one of immediate relevance to the software engineer, and a domain model is a *machine representation* of information about the application-area and the library assets available for the application area. The model can contain general domain information along with data about the form, fit, and function of the available library assets.

This package is part of an evolving series of orientation packages that constitute a comprehensive RLF training program. Eventually, this program will include material for three distinct categories of RLF users:

- end users of RLF-based applications, concentrating on those employing the RLF Graphical Browser (RLF GB);
- maintainers and administrators of RLF-based applications and in particular the underlying knowledge bases on which the applications depend for much of their power;
- RLF model developers and application designers.

During the first part of this modeling orientation, a survey of domain analysis and modeling activities and concepts is presented (see SLIDE 2) with a focus on how these activities fit into a basic system development process framework that is a joint product of the three STARS Prime contractors.

Slide 2

Overview
Basic Modeling Concepts

- Introduction
- Domain Analysis and Modeling Context
- STARS CFRP
- Advantages of an RLF-like approach

During the second part of the orientation, the emphasis shifts to showing how RLF capabilities support the requirements and opportunities afforded by a careful domain analysis (see SLIDE 3) with a focus on how the RLF supports the production and representation of domain models.

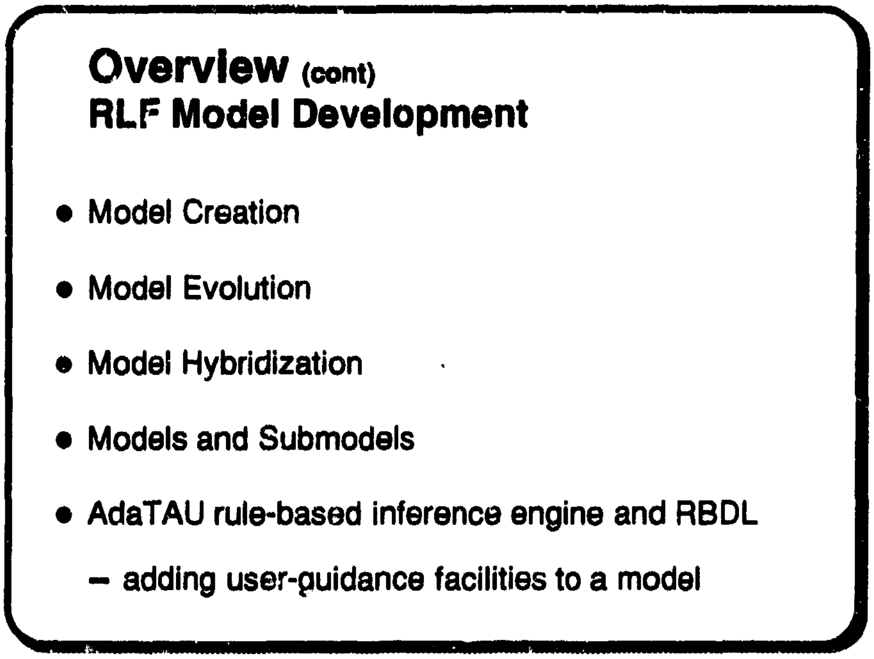
Slide 3

Overview (cont)
RLF Modeling Capabilities

- AdaKNET semantic networks and LMDL
- Hybridization -- attributes and inferencers
- Asset Processing -- defining RLF actions
- Modeling Suggestions

The third part (SLIDE 4) is structured to accompany a demonstration of using the RLF to create, modify and maintain a working library for a simple application domain.

Slide 4



All of the model-building activities will address the problem of how to create and evolve models which focus on the needs of a user community who will interact with a library and the applications that are based on these models.

While a graphical user interface provides library end-users with convenient and powerful methods to access assets and the underlying library model, it is not currently possible to build or modify a library model graphically. Model builders and library maintainers use ordinary text editors to create and modify RLF model description files which are then processed by the respective description language processors.

2 Domain Analysis and Modeling Context

The slides and notes contained in this package address how to develop library models that support the construction of domain-specific reuse libraries. SLIDE 5 lists some reasons to consider a domain-specific modeling approach.

Slide 5

Why Domain Models?

- Widespread belief that reuse is enhanced if done in small, specialized areas
- Software is naturally partitioned into "domains"
- Modeling a software domain structures the collection of assets
- Domain model forms skeleton for library architecture
- Domain information can be used to aid asset retrieval and more general kinds of asset processing

Domain knowledge must be put into the hands of engineers who are building systems in that domain. However, different categories of users may in fact require different views of the information. Domain analysis is a kind of knowledge engineering and a general discussion of this topic is beyond the scope of this orientation package. However domain analysts must have a keen sense of who their user community will be and must organize the information appropriately. The scheme to store and represent domain knowledge should be powerful, yet flexible.

The cornerstone of the RLF approach to supporting reuse is the definition and presentation of supportive domain model information to serve the needs of library users.

Slide 6

What Is *Modeling*?

Modeling is a process of organizing knowledge about a domain (a specific portion of reality/thought)

- **Define The Domain.**

  You must decide exactly what your domain is. You must know or learn something about it.

- **Organize Domain Knowledge.**

  You can know a lot about the domain and still find yourself very challenged. Domain information must be recorded and categorized.

SLIDE 7 indicates some characteristics of models that explain their importance. A model-based approach is needed to support reuse because reuse embodies a shift in the way that software engineering is done. Rather than employing the common mindset that assumes that design is a response to a specific set of requirements, a reuse mindset adopts a balanced problem - product - process point of view. System development takes place with a clear grasp of the available software assets that exist within a specified scope of applicability (a domain). A domain model is a way of articulating the intended scope of reuse for a given set of software assets.

Slide 7

Important Model Characteristics

- **Formal and Representable**

  Models can clarify, abstract, and organize knowledge about a domain.

- **Machine Processable**

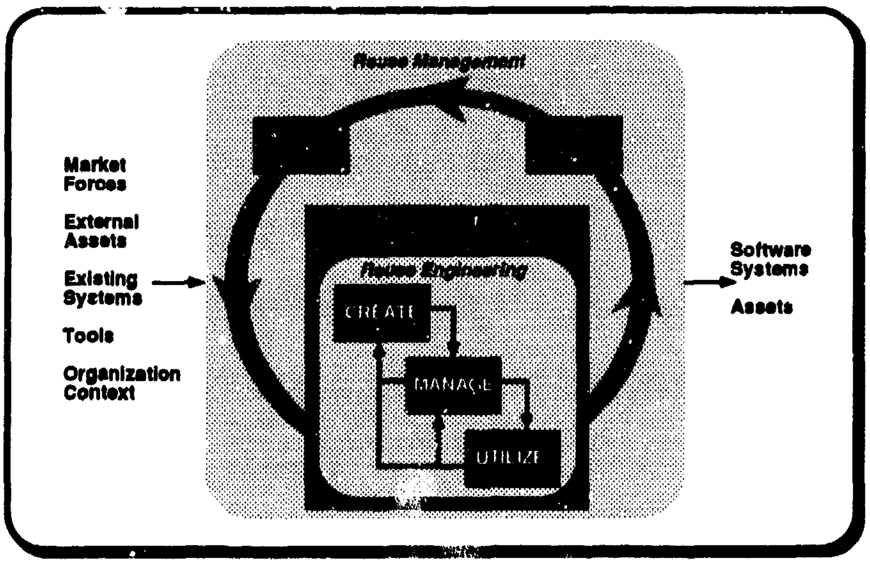
  Tool support enables automation of tasks and Knowledge-based tool support enables reasoning about the domain.

Tool support is necessary because effective reuse will often take place in circumstances that stress a person's intuitive limits in handling complexity. For these situations, modeling techniques must be able to capture the complex semantics of the domain. In particular, sub-system level reuse can be very successful because a greater proportion of an application is built with reused components. Such reuse requires a modeling capability that describes the aggregate collections of components into systems.

3 STARS CFRP

Domain analysis and subsequent modeling are activities that fall within a Reuse-Driven system development strategy that is addressed by the STARS Conceptual Framework for Reuse Processes (CFRP). This framework is not a life-cycle model; rather, the CFRP provides a comprehensive checklist of reuse processes that are meant to work together synergistically and lead to a transformed system development paradigm. SLIDE 8 provides a high-level view of the CFRP and SLIDE 9 lists some of the subprocesses that fall within each of the major categories of the CFRP.

Slide 8



Slide 9

Reuse Management

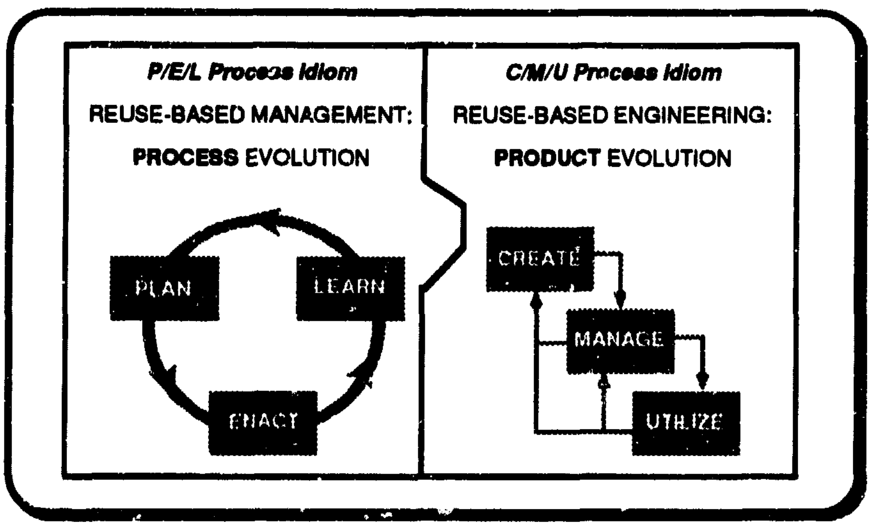
- **Reuse Planning**
  - Assessment
  - Direction Setting
  - Domain Selection
  - Infrastructure Planning
  - Project Planning
- **Reuse Enactment**
  - Project Management
  - Infrastructure Implementation
- **Reuse Learning**
  - Process Observation
  - Process Evaluation
  - Innovation Exploration
  - Enhancement Recommendation

Reuse Engineering

- **Asset Creation**
  - Domain Analysis and Modeling
  - Software Architecture Development
  - Application Generator Development
  - Software Component Development
  - Asset Evolution
- **Asset Management**
  - Asset Acquisition
  - Asset Acceptance
  - Asset Cataloging
  - Asset Certification
  - Asset Metrics Collection
  - Library Operation
  - Library Data Modeling
  - Library Metrics Collection
  - Library Evolution
- **Asset Utilization**
  - Asset Requirements Determination
  - Asset Identification
  - Asset Selection
  - Asset Tailoring
  - Integration of Assets with Application

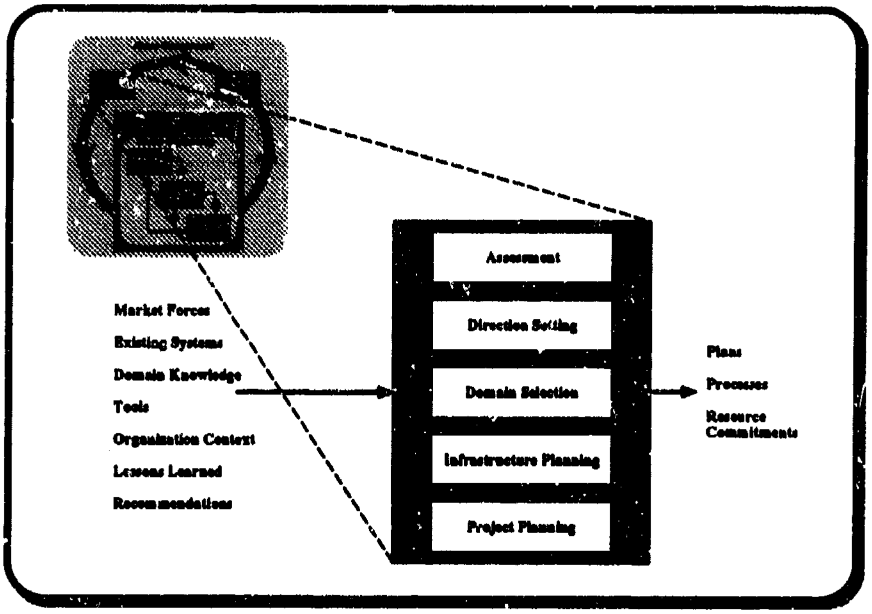
Reuse-based management as captured in the Plan/Enact/Learn idiom and reuse-based engineering as captured in the Create/Manage/Utilize idiom provide both process and product evolution strategies as depicted on SLIDE 10.

Slide 10



With respect to domain analysis and modeling, the Enactment and Planning activities are of particular interest within the CFRP. SLIDE 11 provides an expanded view of the planning activities and shows the resources which are taken into account by these activities as well the products produced by them. For the most part, these are higher-level activities which are beyond the scope of this orientation package. An organization must, however, consider these activities in putting together a reuse plan that will be effective for the organization. To some extent, this plan will be driven by business decisions and will include selection of domains which support those decisions.

Slide 11



In particular, domain analysis intersects broadly with the planning activities of assessment, direction setting, and of course domain selection. Moreover, the creation of a domain model is an important aspect of infrastructure planning. Domain selection is an important first step to conducting a successful domain analysis. Domain selection should take place in an expanded reuse planning context. In fact, domain selection is as much a management/business/strategic decision as it is an engineering decision.

There are many processes that specifically address domain analysis within the area of reuse planning. Some of them are summarized in SLIDE 12.

Slide 12

Key DA Planning Activities

- Reuse Assessment
  - Identify Organization Domains
  - Characterize Domains
- Direction-Setting

  Establish DA Objectives
- Domain Selection and Planning
  - Infrastructure Planning
    - * Select DA Modeling Techniques
    - * Select Domain Engineering Technology
  - Project Planning

    Configure Asset Creation Projects

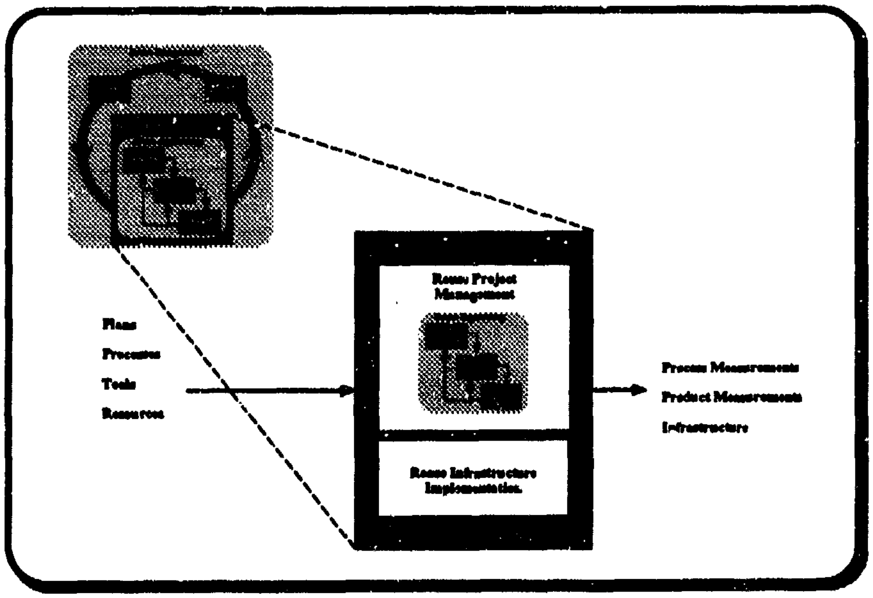
The following bullets expand on the issues raised on the previous slide.

- Reusc Assessment
  - Organizational Assessment – ties into Reusability Analysis
  - Domain Identification and Characterization – conducted in a different context from Organization Characterization
- Direction-Setting
  - Domain Analysis Objectives – in light of
    - * Overall organizational objectives
    - * Organizational Reuse Objectives
  - Domain Metrics/Criteria for Success
  - Select Domain Analysis Methodology
- Domain Selection – Different organizational contexts for domain selection criteria
  - Technical criteria
    - * Stability of domain technology

    * Maturity (e.g. number of systems implemented and length of time fielded)
    * commonality across applications
    * performance constraints
    * flexibility of user requirements to accommodate reused resources
- Organizational criteria
    * Strategic interest in domain
    * Availability of expertise
    * Ability to form domain engineering team for this domain (i.e., how much does it cut across current organizational structure?)

Similarly, SLIDE 13 expands upon the enactment process. Domain analysis and modeling enable construction of valuable reuse-support structures which aid a successful reuse-based approach to system development. Not only are assets created, managed and used in the development of a new system, but the knowledge about these assets and the relationships among them also undergoes an analogous kind of Create/Manage/Use engineering activity.

Slide 13



Domain analysts build models. These models help engineers understand the application domain and any existing assets that can be applied to constructing systems within the domain. The models are themselves valuable assets that must be maintained and modified (i.e. managed). Asset managers may need to maintain multiple models and update them as required. Users (i.e. Application engineers) make use of the models in making important decisions about how to develop solutions to the problems being addressed by the system under development. Thus, not only are the assets used, but the models are used as well to understand how to make effective use of the assets. SLIDE 14 lists some of the relevant activities.

Slide 14

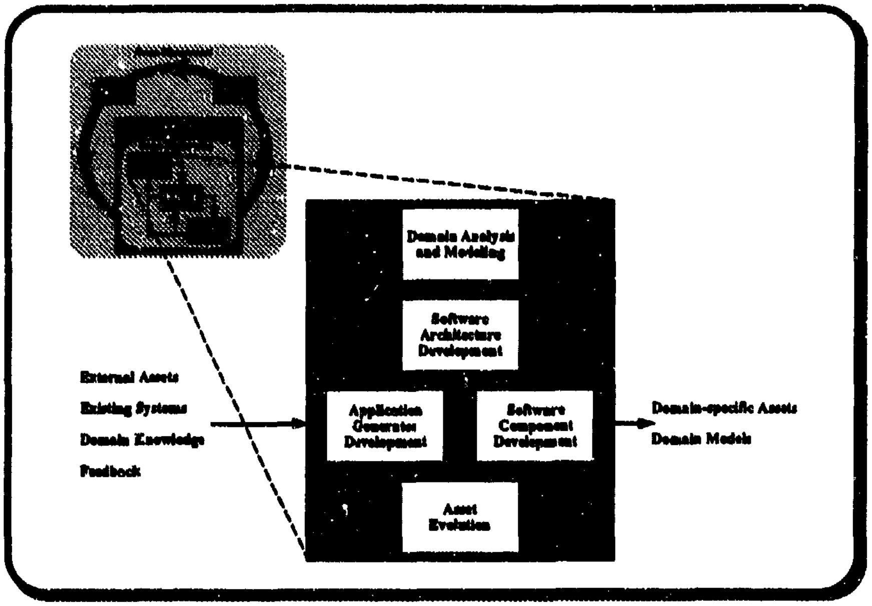
C/M/U and Domain Analysis

- Asset Creator
  - domain analyst
    - builds domain models
  - domain engineer
    - builds components
    - adds/classifies components w/in domain model
    - validates components wrt model
- Asset Manager
  - Asset Qualifier/Certifier
  - Asset Cataloguer
  - Library Model Maintenance/Evolution
  - Usage History Monitor
  - Promotion of Assets/Commissioning New Assets
- Asset Utilizer

With respect to the RLF, asset creators make and modify RLF models while asset managers administer and maintain them. Asset users browse and query RLF models and invoke applications which make use of them. However, asset creators will also find themselves in the position of model users (e.g. obtaining, incorporating and tailoring already-built models of subdomains) while asset users may help create modified RLF models (e.g improved rule-based heuristics, or identify gaps in model coverage and asset availability).

The creation activity is broken out further in SLIDE 15. In particular, domain analysis can lead to key decisions about how assets can be created (e.g. composition vs. generation) and an on-going effort to improve domain information can lead to transitions between forms of asset support – e.g. from custom construction of subsystems, to (semi-)automatic composition of systems from available components to parameterized generation of components and even entire systems. Not only are assets undergoing an evolutionary process, but so to are the models.

Slide 15



Note that architecture development is distinct from domain modeling per se. A domain model represents commonality and differences within the "problem space" of the domain (system features), whereas the architecture represents alternatives in the "solution space". Architectures that are supportive of reuse differ from ordinary system architectures in that they might include generative components as well as constructive.

There a number of domain analysis activities within the **Create** category of the **Enact** idiom. These activities produce a wide variety of domain engineering products. These activities are summarized on SLIDE 16 and the products produced by the activities are enumerated in the list following the slide.

Slide 16

Key Domain Analysis Activities

- Domain Definition
- Domain Scoping
- Domain Information Gathering
- Domain Modeling
- Domain Model Validation
- Domain Refinement
- Domain Engineering

The following bullets expand on the activities listed on the previous slide and show some possible resulting products. The last item on the list, domain engineering, corresponds to "architecture development" identified on SLIDE 15.

- Domain Definition
  - Domain Definition Statement (intensional)
  - Domain Exemplar Set, chosen from a range of systems
    * Implemented/fielded systems
    * Requirements for new systems
    * Market studies – forecasts
  - Domain Genealogy – Different historical relations between systems
    * direct successor system
    * leveraged system
    * requirements reuse (black-box)
    * independently developed
    * competitively developed
- Domain Scoping
  - Domain Interconnection Model
    * Operational Relations – e.g., Domains related through operational interfaces in end-user system or through implementation relationships

    * Specialization/generalization relations – e.g., outlining programs relate-to hypertext programs; these domains are related not because one is used to implement the other, or because they both co-exist and may even exchange data in a single environment, but because one represents a superset of the functionality of the other.
    * Analogy domains
    * Life cycle domains
  - Domain Boundary Decision Report

- Domain Information gathering leads to a Domain Information Inventory[1]
- Domain Modeling
  - Domain lexicon
  - Requirements-Oriented Feature Model
  - Models of environmental characteristics – often much reuse potential hidden
  - Feature binding time model
  - Error semantics model
  - Feature Modeling Principles – Models can begin informally; as they scale up, their semantics must become clearer; inheritance becomes more useful; feature model should flag any functions not included in exemplar set
- Domain Model Validation – done along with information gathering, modeler should verify terminology is in keeping with the application domain
- Domain Refinement
  - Articulate domain boundaries
  - Defer modeling of sub-domains
  - Clarify individuals in each domain model
- Domain engineering – must integrate results of domain analysis with requirements analysis; also, DA results stand on their own

The Domain Modeling activity in SLIDE 16 is most directly related to the core material of this orientation. There are many kinds of models which can be produced during this activity, including several for which the RLF provides a suitable representation medium. Some of these models are on SLIDE 17.

[1]Techniques of information-gathering include searching databases, networks, technical communities ethnography, participants observation of a single engineer's grasp of problems in the domain, group interviewing

Slide 17

Domain Modeling Products

- Domain Lexicon
- Requirements-Oriented Feature Model
- Models of environmental characteristics
- Feature binding time model
- Error semantics model

A semantic network model is sufficiently general and powerful to capture all of the model types in the previous list.

4 Advantages of an RLF-like approach

As is evident from the previous discussion, a reuse- and models-based approach to system engineering requires significant tool and infrastructure support. There are a spectrum of techniques to apply to domain modeling and domain model representation. These approaches range from semantically weak ones such as keywords to semantically strong ones such as an object-oriented (O-O) approach. A list of candidate approaches is given on the next slide.

Slide 18

Domain Model Approaches

- Keywords
- Faceted Classification
- Structured Inheritance Network
- Object-Oriented

There are two main competing approaches that have received widespread use: faceted classification (which is a structured form of keyword-based techniques) and object-oriented (O-O) classification. In fact, RLF models can be built which provide all of the features of a faceted scheme. Moreover RLF shares several features with O-O techniques including support for inheritance of properties (including multiple inheritance) and the specification of class (or category) attributes.

The RLF with its semantic network, rule-based, and hybrid modeling techniques is well suited to provide modeling support. It represents a semantically rich modeling medium which includes most of the O-O features that are important to reuse, and leaves aside other features that are not as useful for reuse. Some reasons to consider the RLF are summarized in SLIDE 19.

Slide 19

Why Use the RLF?

- includes all key features of Faceted Classification

  facets and facet terms can be provided through RLF relationships or fact base schemas

- shares key features with O-O modeling including (multiple) inheritance, but
- doesn't require O-O design or programming approach
- separates model from application which uses the model
  - different applications run over same model
  - different models accessed by same application
- models maintain variants of subsystems at any level
- enforces semantic restrictions
- models as first-class objects, artifacts, assets

However, RLF models are not just O-O models, and in fact can be built for systems that are designed and implemented without assuming an O-O programming paradigm. Whereas O-O techniques encapsulate behavior with classes and objects, this association may not be appropriate for general reuse libraries. In fact, evidence suggests that "code inheritance" may not be an effective way to support reuse. RLF models empower applications which use the model in application-specific ways – there are no built-in assumptions about how systems built from the model are constructed. One can view an RLF model as a rich data structure and RLF abstract data types provide the necessary interfaces to interact with and manipulate the data structure.

RLF library models have a subset of O-O features that are most appropriate to support reuse. The use of RLF semantic network models enables effective modeling of subsystem variants at all levels within the model, and not just at the frontier portion of the model.

Domain modeling is a form of knowledge capture and a knowledge representation formalism is required to encode and manipulate the captured information. The RLF provides two complementary forms of knowledge representation. In the following sequence of slides, the semantic network information subsystem called AdaKNET will be surveyed.

AdaKNET provides the basic supporting structure for an RLF library domain model. It

- encodes static aspects of a domain model
- provides structure for library
- describes reusable assets and their properties and relationships
- has scalable power of representation

Slide 20

Knowledge Representation

- System of formal conventions for encoding knowledge -- the modeling vehicle
- The RLF uses AdaKNET as part of its Knowledge Representation Scheme (KRS)
  - -- based on KL-One (Brachman)
- AdaTAU is an auxiliary KRS
  - -- Other auxiliary KRS have been used with AdaKNET including CLIPS

AdaTAU contributes the following capabilities to the RLF:

- as an auxiliary modeling formalism, it supports those domains which require a less formal representation
- can directly encode faceted classification schemes
- can "grow" faceted schemes into more taxonomic models over time
- handles a simple interactive protocol for eliciting and processing less structured information within the system
- provides means to reason about the model represented using AdaKNET

5 AdaKNET Semantic Networks and LMDL

Before considering the features of AdaKNET in depth, SLIDE 21 notes the distinction between the implementation of AdaKNET and the general modeling notation and examples presented in the rest of this section. Much of the discussion in this section would apply to any general semantic network system, including KL-One. The actual use of AdaKNET and LMDL (Library Model Description Language) is covered in depth later in this orientation.

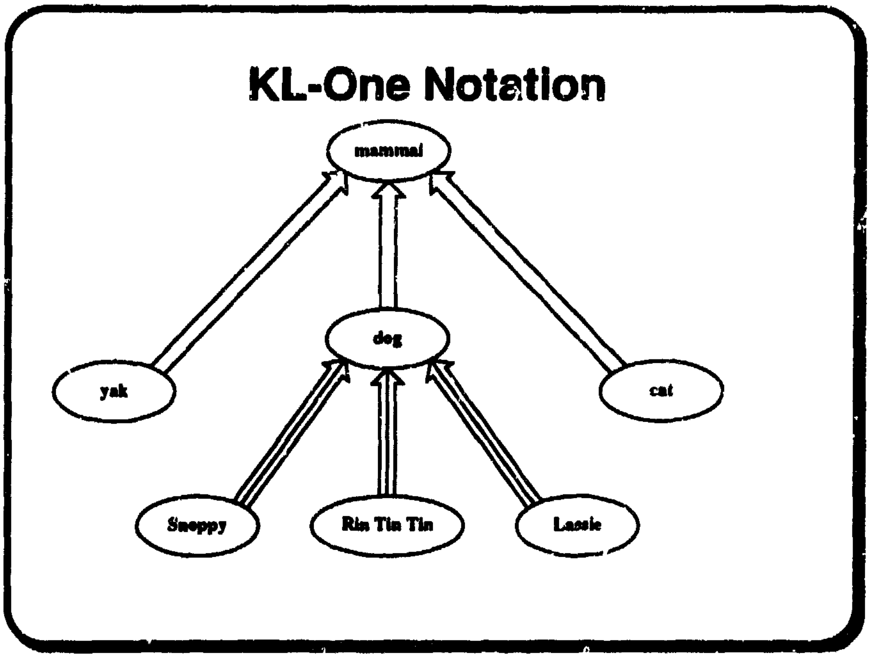
Slide 21

AdaKNET Principles and Notation

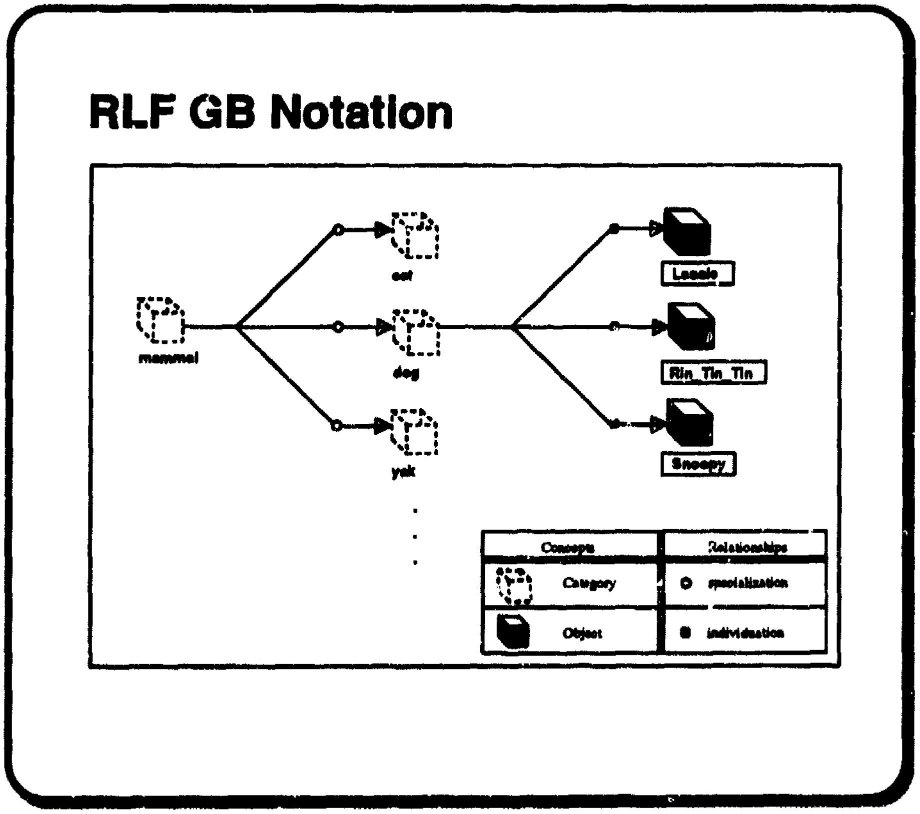
- Semantic network principles are introduced first
- Graphical notation is reused from original KL-One treatment.
  - This notation is not supported in RLF tools
  - RLF networks are specified textually and translated to machine representation
  - RLF Graphical Browser displays networks as trees with replicated nodes
- Designers can use sketches like those presented here to design models incrementally
- AdaKNET graphical model editing tool is a potential future enhancement

As an indication of the different representations of network structures used in this tutorial, the next three slides show a KL-One style view, an RLF_GB style view, and an actual LMDL network textual specification for a simple network model. Both the KL-One and LMDL notations are used extensively in the tutorial. The RLF_GB notation conventions are treated more completely in the RLF User Tutorial.

Slide 22



Slide 23



LMDL Notation

```
library model "Mammal Model" is

   root category mammal is
   end root category;

   category yak (mammal) is
   end category;

   category dog (mammal) is
   end category;

   category cat (mammal) is
   end category;

   object "Lassie" (dog) is
   end object;

   object "Rin Tin Tin" (dog) is
   end object;

   object "Snoopy" (dog) is
   end object;

end "Mammal Model";
```

Slide 24

Semantic networks are also known as structured inheritance networks (SLIDE 25) because information located at one node of the network is inherited by those nodes of the network which are descendants of that node. In particular, links to other parts of the network established for a parent node are inherited by child nodes, although those links can be made more specific (constrained) as examples to follow will show.

Slide 25

AdaKNET

- AdaKNET provides a language formalism -- LMDL

  A formal language enables the clear and precise definition of model features

- AdaKNET is a structured inheritance network

  Such networks provide an economical and practical representation of category and object relationships

There are five main features which determine the ability of AdaKNET to represent knowledge which are shown in SLIDE 26. These are discussed in the following slides. In the course of explaining AdaKNET modeling, various model diagrams will be drawn to help illustrate AdaKNET features. The next slide summarizes the basic capabilities of AdaKNET.

Slide 26

Basic Features of AdaKNET

- Categories
- Objects
- Individuation

  Category Membership

- Specialization

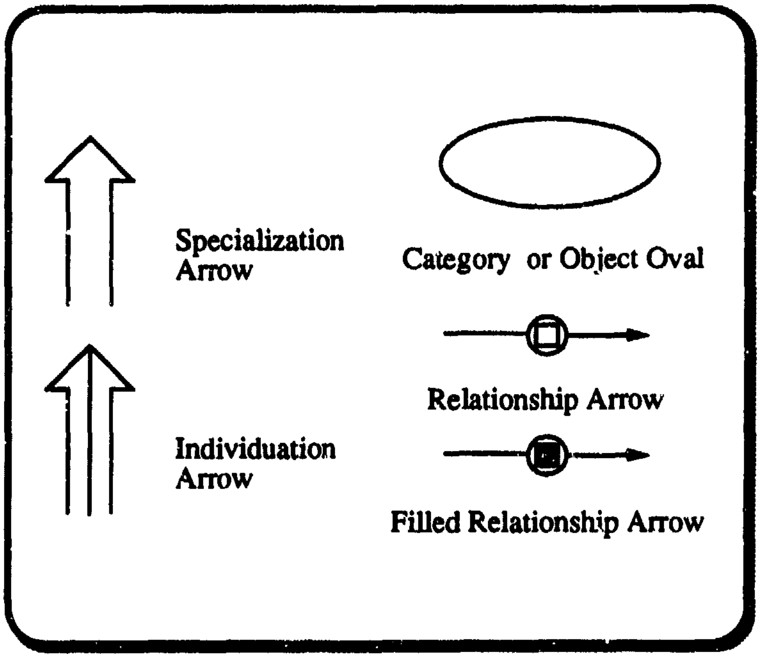
  Specific-General (IS-A) relationships

- Aggregation

  Whole-Part (HAS-A) relationships

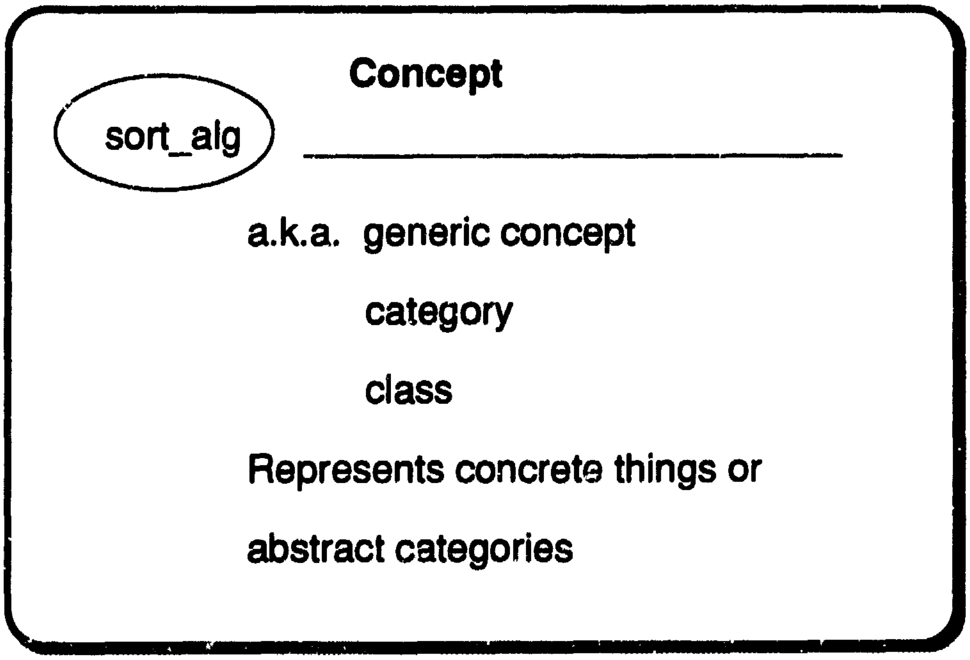
SLIDE 27 illustrates the notational conventions used in these diagrams – they are similar to those used in the general literature to present and discuss semantic networks.

Slide 27



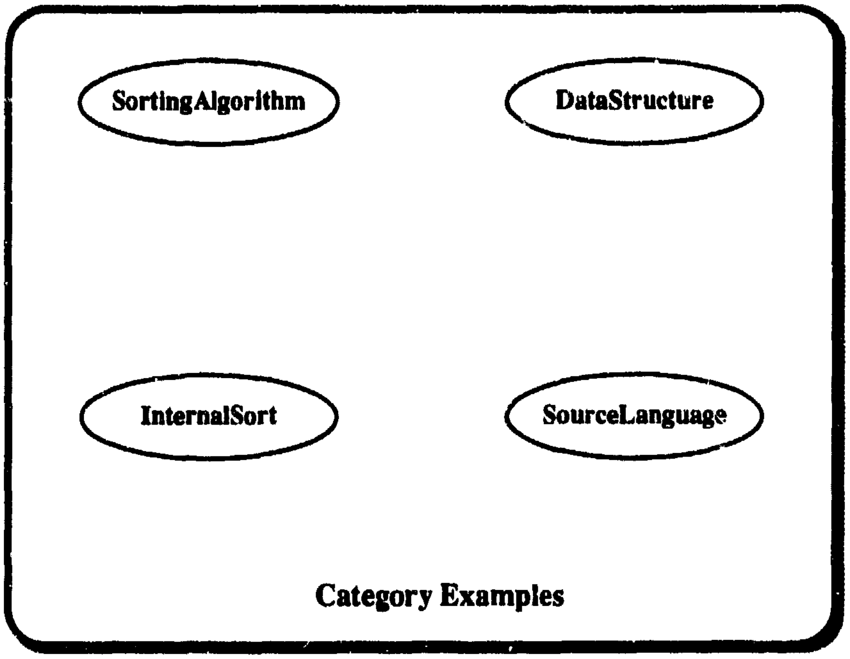
Categories (a.k.a. concepts) provide the means to model general classes or kinds of things.

Slide 28



The following slides show some categories that are appropriate for a model of the sorting domain.

Slide 29



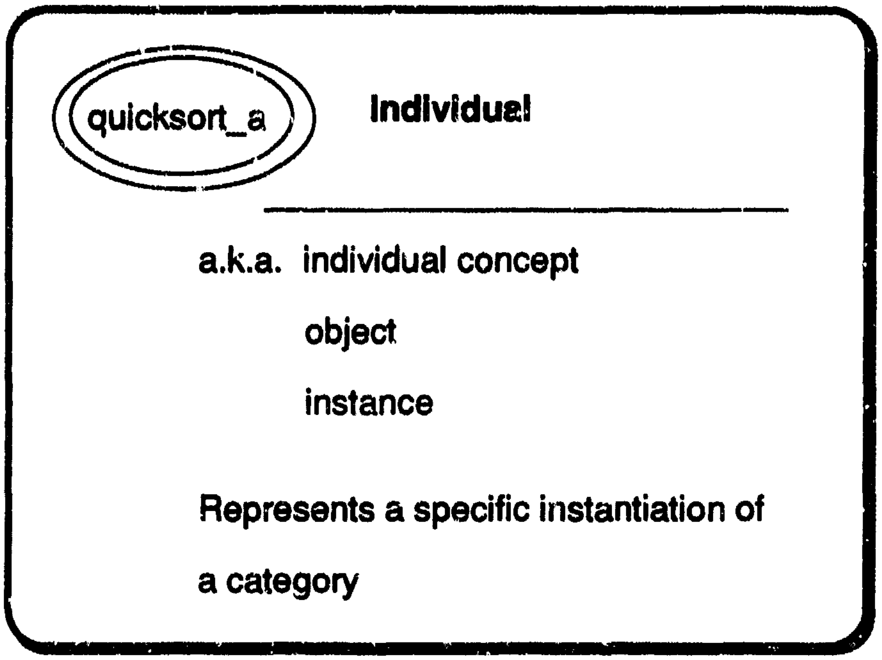
Slide 30

Exercise 1

Create some concepts for the domain of ...

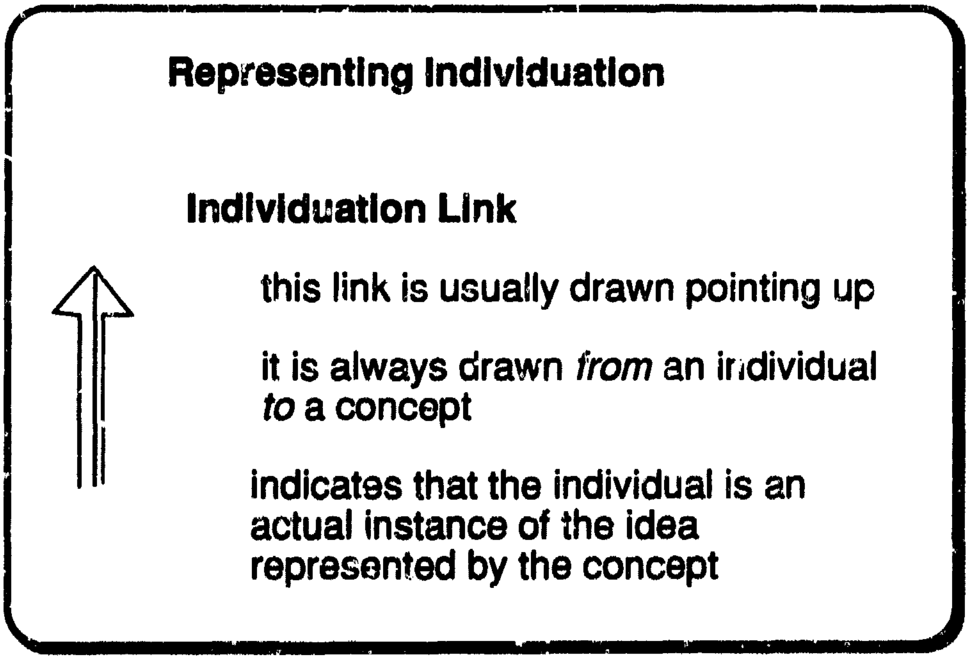
Objects (a.k.a. instances or individuals) model category instances – particular things.

Slide 31



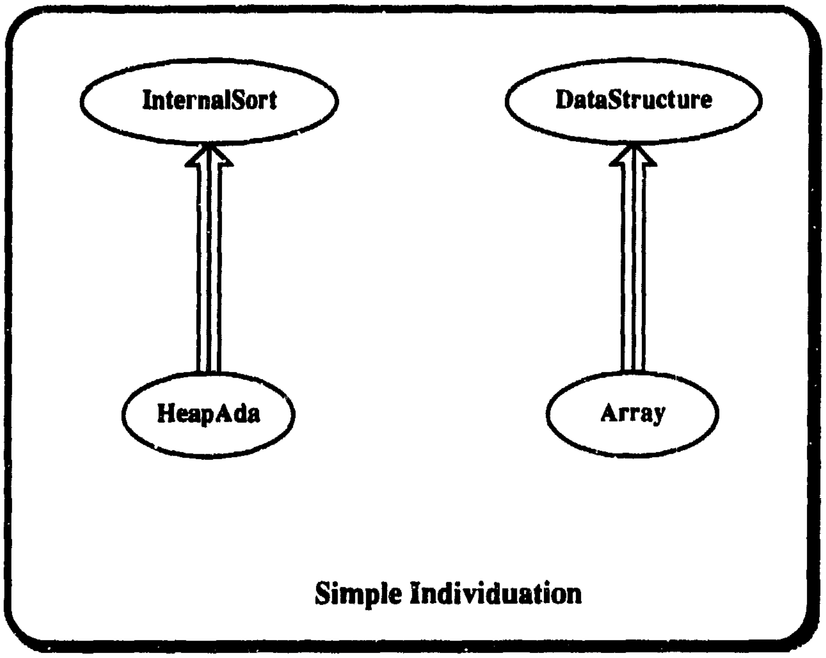
Object membership in a category is represented by an "individuation" link. Objects are sometimes referred to as individuals. Objects receive their essential identity from their relationship to the containing category.

Slide 32



The next slide shows two examples of objects linked to their parent category.

Slide 33



Note that it is not always clear when an idea (an intellectual concept) is best represented as a category or as an object in a semantic network. There are situations where both may seem to be possible and the circumstances surrounding the application of the model may tip the balance in favor of one or the other. For example, **Array** is represented as an individual on the previous slide; in some situations **Array** may be best represented as a generic concept (category) with many specializations. For reuse library applications, SLIDE 34 suggests a generally useful viewpoint on distinguishing generic concepts and individual concepts.

Slide 34

Categories vs. Objects

- A generic concept represents a category of assets or information type used to distinguish assets
- An individual concept (object) represents a particular asset or elemental piece of information about assets

Slide 35

Exercise 2

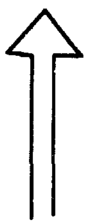
Create some objects for the categories you created in exercise 1 in the domain of ...

Category/sub-category relationships are captured as specialization links in AdaKNET networks.

Slide 36

Representing Specialization Relationships

Specialization Link



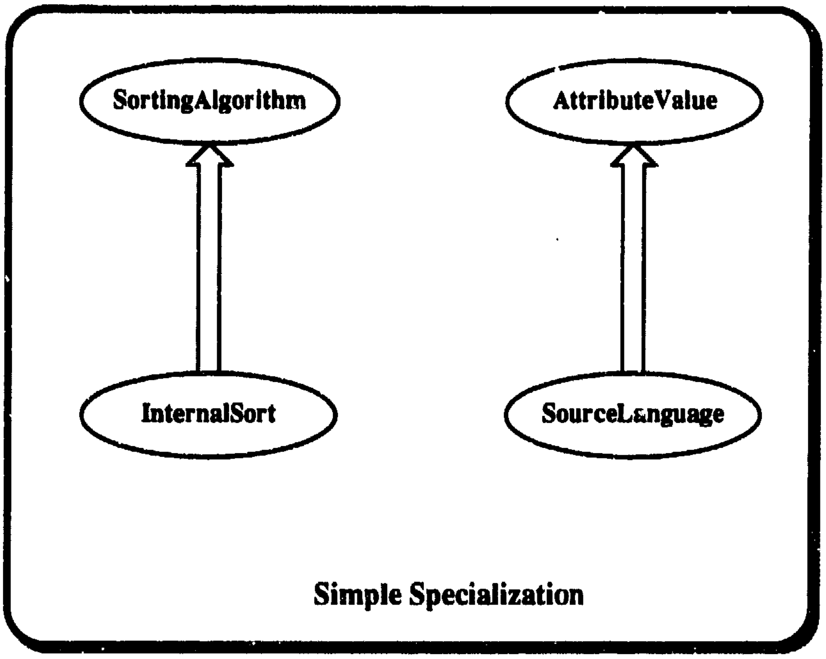
a.k.a. 'is-a' link

indicates the "lower" concept is a narrowing of the category represented by the "higher" concept

link is usually drawn pointing up

it is only drawn between concepts

Slide 37



Slide 38

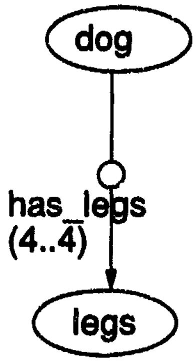
Exercise 3

Create some specializations of the categories you created in exercise 1 in the domain of ...

Properties of categories are provided through aggregation links. These links allow category relationships to be expressed that provide whole-part information or further define category characteristics. Adaknet roles (as these aggregation links are called) have three distinctive parts as shown on the following slides.

Slide 39

Representing Aggregation Relationships



a.k.a. 'has-a' links, roles, rolesets, relationships

Usually represents either a part-subpart relationship (e.g. "car has-a engine")

OR

a characteristic (e.g. "car has-a color")

Each role is identified by a name, a range (the number or times the role can be repeated for an individual) and a type which identifies the nature of the role.

Slide 40

Representing Aggregation Relationships (cont)

Aggregation links

3 parts:

name	has-leg
range	(4 .. 4)
type	leg

An RLF model designer should pick useful and informative names for aggregation links.

Slide 41

Representing Aggregation Relationships (cont)

name - generally a verb which indicates the nature of the relationship

The designer must also establish bounds on the number of anticipated repetitions of each role as well as the basic kind (type) of information being captured by the role.

Slide 42

Representing Aggregation Relationships (cont)

range - a numeric range indicating how many copies of the relationship may exist simultaneously

Range is an ordered pair:
(min.,max.)

e.g. (1..1) (0..infinity) (4..5)

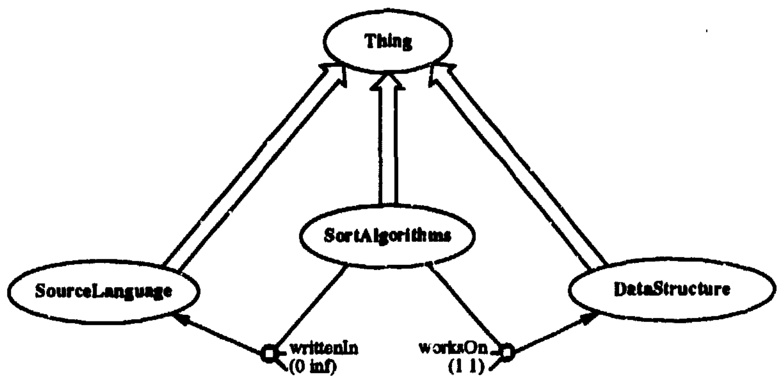
Slide 43

Representing Aggregation Relationships (cont)

type - generally a concept which specifies what kinds of things can be used as the value of this aggregation relationship

The next slide shows a simple example of the definition of relationships provided by roles.

Slide 44



Simple Aggregation

Slide 45

Exercise 4

Create some aggregation links for the specialization hierarchy you have created in the previous 3 exercises in the domain of ... At least one of these links should represent a whole-part relationship and at least one should model a characteristic relationship.

The ability of a structured inheritance network to model domain information is actually contained in the specialization (category to subcategory) and aggregation hierarchies which form the infrastructure of the network. Aggregation information is inherited along the specialization hierarchy. Objects that instantiate a class require other objects that "fill" the corresponding aggregation slots possessed by the category to which they belong. These aggregation instances are called role "fillers." In general, aggregation information can be localized (restricted and decomposed) at lower levels of the specialization hierarchy. This process is covered in the next several slides.

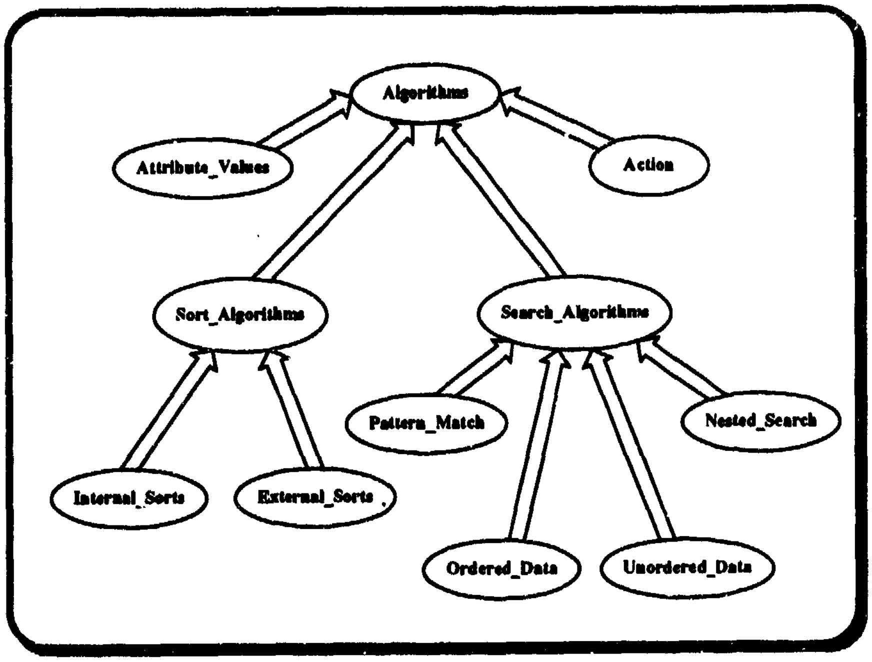
Slide 46

The Specialization Hierarchy

- Fundamental model properties are conveyed through AdaKNET's specialization hierarchy
  - provides taxonomy, conceptual decomposition, subsumption hierarchy
  - categories are "declared" in the context of this hierarchy
- **Subsume** (*Oversume*)

  To classify in a more comprehensive category or under a general principle

Slide 47



Aggregation information is inherited along the specialization hierarchy.

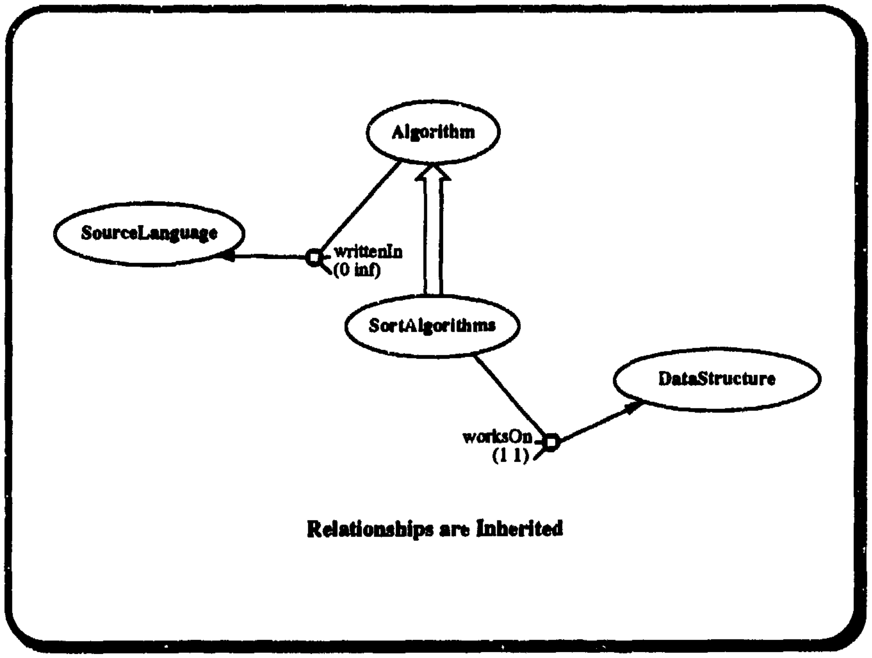
Slide 48

Inheritance

- Whenever a category that "has aggregation" is specialized, that aggregation structure is inherited by the more-specific ("lower level") category.
- Roles are installed at the *highest* level appropriate.
- You can add new, local roles to any category, regardless of whether it has inherited roles.
- Objects in category possess all roles of containing category, whether locally defined or inherited.

SLIDE 49 provides a simple example where a relationship in a parent category is inherited by a subcategory and the subcategory has its own local relationship.

Slide 49



Slide 50

Exercise 5

For the model fragment you have been developing in the previous exercises, install new roles if necessary and then show at least category with a role which is inherited from a parent category as well as role declared local to the category.

In fact, AdaKNET permits a limited form of multiple inheritance.

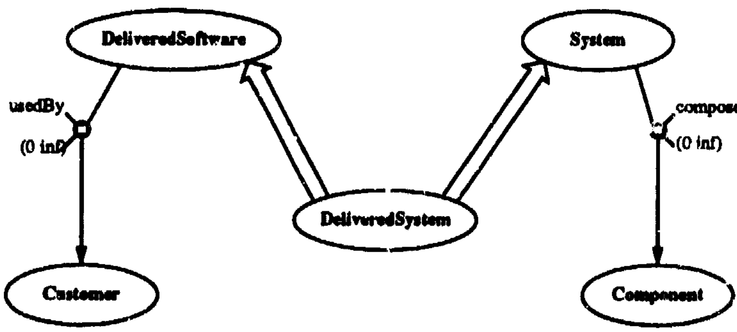
Slide 51

Multiple Inheritance

- AdaKNET supports multiple inheritance -- categories can have more than one parent category
- Usually this means that the specializing category inherits the relationships of all its parents, but:
  - parents have common ancestor, thus common relationships
    The intersection of the available relationship restrictions from both parents are imposed
  - parents have different relationships with same name
    This situation is not allowed in AdaKNET

The next slide shows a simple example of multiple inheritance where the inherited relationships are distinct. Note that multiple inheritance is often useful in AdaKNET models because different access paths to an object or category may support different classes of users. In the RLF, an object may be declared to have multiple parents – it is not necessary to declare an intermediate category for multiple inheritance to occur.

Slide 52



Multiple Inheritance

When a relationship is simultaneously inherited from multiple categories, it is necessary to merge the range and value restrictions of the relationship so that it continues to describe both parent relationships. The relationship's range must be the largest possible range that falls within all the parents' ranges for the relationship. Similarly, the relationship's type must be the same as or subsumed by all of the parents' types for the relationship. If a range or type meeting these criteria does not exist, the inheritance is not possible without violating subsumption, and the specialization is not allowed. The library model specification translator will report an error when constructing the library model representation and abort.

Any parent relationships which have the same name but do not descend from a common ancestor are distinct relationships. In order for these relationships to be inherited by a single concept, the name conflict must be resolved by renaming one of the relationships before the child concept can be created. This should be done in the LMDL specification for the library model.

Users just beginning to create models in RLF should carefully evaluate the need for multiple inheritance. Multiple inheritance is not used in the rest of this orientation.

As role information is inherited, it can be made more specific, but it can never be generalized. Roles can be narrowed in terms of cardinality (the number of permitted repetitions) and role type – at a subcategory, there may be fewer repetitions and/or the type of role filler may be restricted to a subtype of the original role type. These strict subsumption semantics for AdaKNET distinguish it strongly from most O-O approaches.

Slide 53

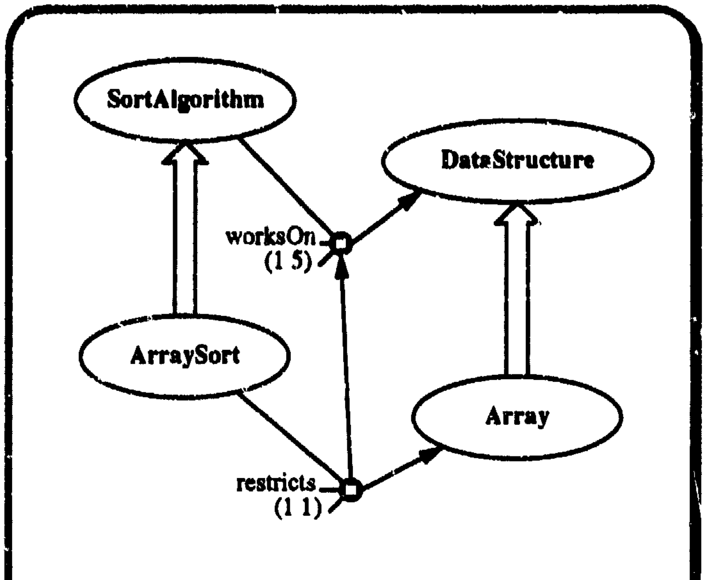
Narrowing Roles

- Inherited roles can be left unmodified or they can be *narrowed*.
- Two ways to narrow an inherited role:
  - Narrow the type

    new type **must** be a specialization of the original type
  - Narrow the range
    - You can decrease the range, not increase it
    - You can converge the range, i.e. make max. = min.

The next slide shows a simple example of relationship restriction where the relationship is restricted both by type and by range.

Slide 54



Narrowing Relationships

Slide 55

Exercise 6

For the model fragment you have been developing in the previous exercises, show examples of role restriction that narrow the type and the range of an inherited role.

Role differentiation is a kind of *specialization* for roles:

- subsetting - splitting off a part of a role (but perhaps leaving some parts of the original role alone), and
- partitioning - splitting a role completely into disjoint parts

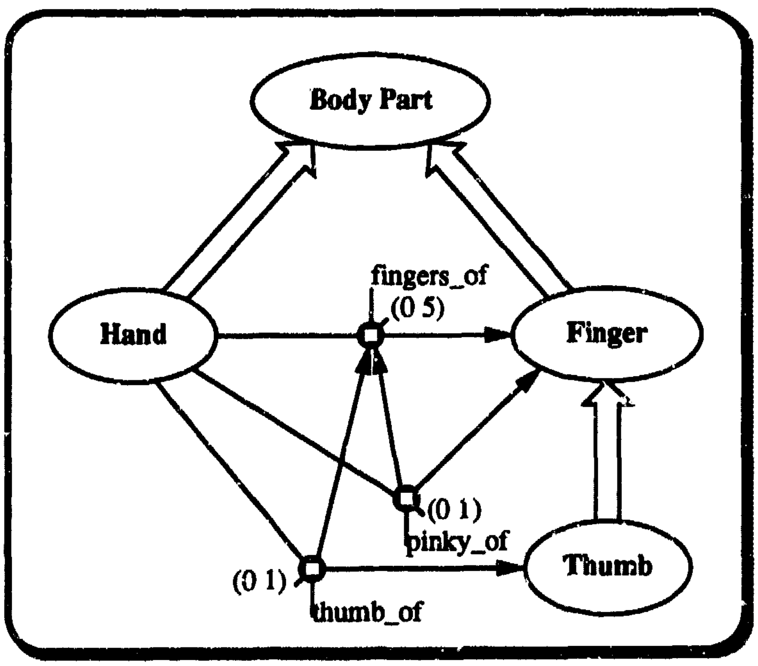
Slide 56

Role Differentiation

- Differentiation allows you to divide a role into a set of subroles.
- Two ways to differentiate a role:
  - Partitioning
    role divided into a set of subroles and each filler must be assigned to exactly one subrole (no further role differentiation possible)
  - Subsetting
    role divided into a set of subroles and each filler can be associated with zero or more of them (subset roles can be further differentiated)

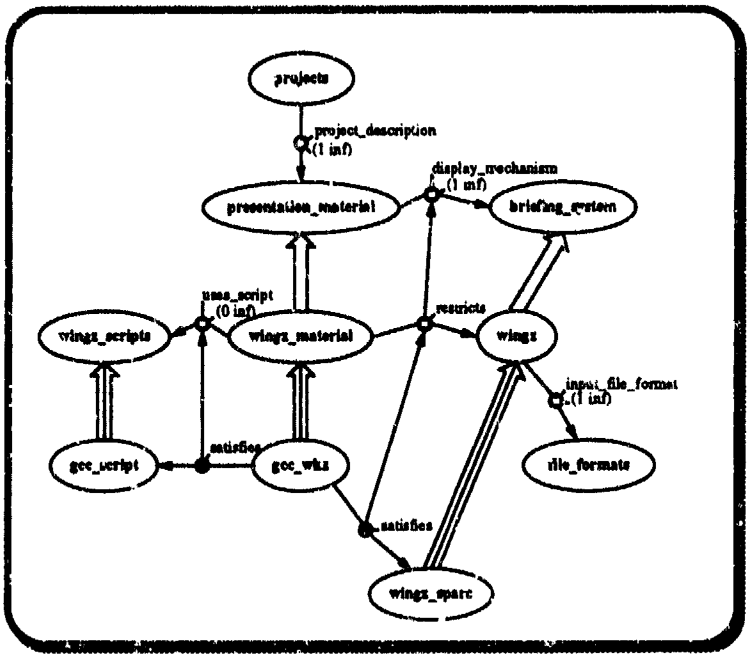
The next slide shows a small example of relationship subsetting where a relationship is decomposed into two named relationships which do not partition the original relationship. This orientation will not consider differentiated relationships further as they are more appropriate for complex modeling situations. The RLF Modeler's Manual provides should be consulted for more information.

Slide 57



The following slide is a small but representative example of a model built using AdaKNET principles. In the hands-on session to follow, the development of a software model will be undertaken that further illustrates the principles discussed above. Note that in this and subsequent network illustrations, objects are not drawn with double ellipses enclosing them since the use of the lined arrow for an individuation link normally makes in clear that the node represents an individual class member and not a subclass.

Slide 58



Actual RLF networks are created from network description files in a specification language called LMDL - Library Model Description Language.

Slide 59

Library Model Description Language – LMDL

- Ada-like non-procedural language
- Specifies all objects and relationships in the model
- Is translated to produce files which represent the model
- Is modified and re-translated to change the model

A survey of the usage and features of LMDL is presented in the third major part of this tutorial.

6 Model Hybridization

The actual contents of an RLF library are located relative to categories modeled in an RLF network. But the storage of these contents (typically as file objects) and the processing and manipulation of these contents are provided by additional RLF features layered on top of the network. One simple kind of asset processing is the simple display of text file assets in a window on the library user's display screen.

Another kind of asset and library model processing is provided by the AdaTAU inferencing system which is also layered on top of a basic network model. Inference bases can be used to set up and control complex library interactions that are themselves examples of sophisticated asset processing. One example of such processing is system configuration whereby a subsystem is assembled, at least semi-automatically, from assets that are individually stored in the library, possibly after the assets are tailored to user requirements elicited during the inference process.

The attachment of AdaTAU inference components, the location of asset files, the storage of additional asset state information (e.g. numeric or text-string values and longer textual class or object descriptions stored in files) and the specification of RLF "actions" to process assets on an individual basis are all currently provided as extensions to the LMDL language.

This additional layering, and the binding of AdaTAU inferencers to particular AdaKNET classes and objects, is called "hybridization." Hybridization allows a text item, an integer item, a file item, or an inference base to be associated with an AdaKNET category or object. Moreover, RLF actions allow the internally provided RLF "inferencing" action to be augmented by other large-grained asset processing services.

Slide 60

RLF Hybridization

- Allow a text item (string or file), an integer item, a general file item, or an inference base to be associated with an AdaKNET class or object.
- The actual file for a software component can be bound as a text file to an object.
- RLF graphical browser allows for viewing of associated files (and in general for execution of RLF actions).
- Allows inferencer support to be context-sensitive.
- Switches between inference bases make moves through library.

RLF attributes are used to annotate categories and objects with static information that provides additional data about them. Attributes are the means by which actual file contents are attached to reusable assets which are modeled as objects in an RLF model. SLIDE 61 illustrates some basic features of attributes.

Slide 61

Attributes

- can be integers, strings of characters, or files
- have names so they can be referenced and used by RLF *actions*
- file attributes can be viewed, extracted, or otherwise manipulated
- attributes are not inherited -- they must defined at each category or object where they are useful

The next slide introduces a companion knowledge representation system to AdaKNET. This subsystem is called AdaTAU (TAU stands for Think - Ask - Update) which is the basic process by which new facts are deduced from a current set of facts through a process of rule selection and execution. AdaTAU features and basic operation are elaborated on later in this package. The main usage of AdaTAU has been to "program" a user's interaction with a domain model and to direct the user to the completion of specific tasks that the domain model supports. Possible uses include assisting a user to:

- navigate an AdaKNET model
- compose a system from objects in an RLF-based library
- enter and qualify new objects within an RLF-based library.

In addition, other kinds of inferencing support have been added to models defined using AdaKNET. NASA's CLIPS inferencing system has been integrated with AdaKNET by the CARDS program to support system composition from library components.

Slide 62

AdaTAU

- Rule-based inferencing engine
- Rule bases defined in RBDL -- Rule Base Description Language
- Inferencing can be distributed in different areas (e.g. nodes of the network)
- Captures per-usage, dynamic, non-structural knowledge
- Tailors advice to the orientation and goals of user
- Interacts with the user to gather per-usage information
- Based on responses, make deductions about model elements and contents that are of interest to the user
- Can focus user to specific category or object
- Requires additional knowledge modeling

7 Asset Processing

In addition to the built-in notion of processing inference bases via AdaTAU, the RLF permits the modeler to introduce either internal or external model processing elements called **Actions**. Some basic facts about actions are listed on SLIDE 63. Actions let the user process categories and objects and permit access to system and library resources within a context appropriate to specific categories and objects. The context for these actions is provided by RLF attributes. In particular, RLF actions provide the mechanism by which library assets can be processed. Two simple kinds of asset processing are asset viewing and asset extraction.

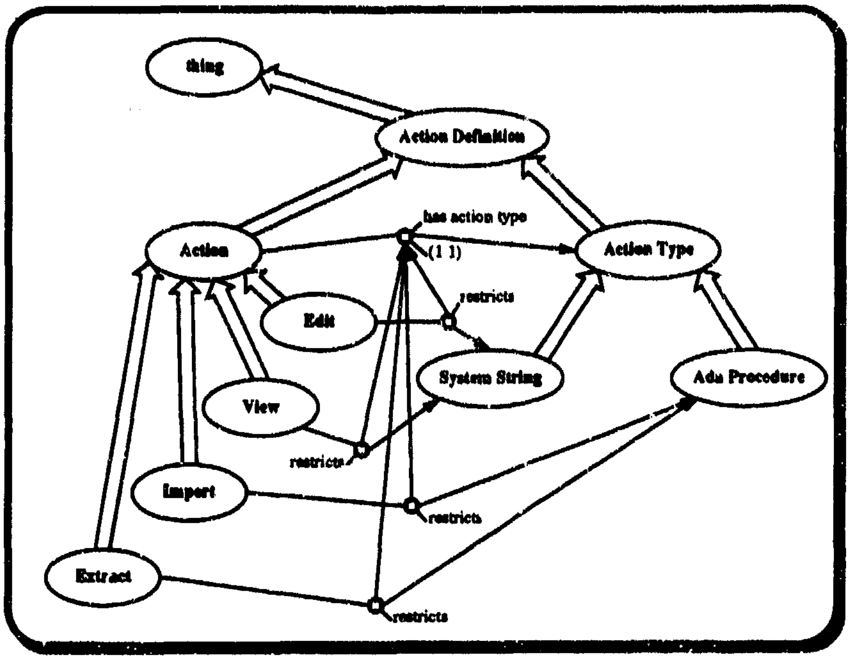
Slide 63

Actions

- provide library users with appropriate system and library services
- basis for asset viewing and extraction
- defined within LMDL specifications -- sample Action sub-model provided in RLF distribution
- implemented as system calls, or internal Ada procedures
- inherited like relationships
- have names, an action category, a list of action targets, and a list of action agents
  - targets reference attributes which provide input for the action
  - agents reference attributes which affect how the action is performed
- can be privileged which means they are unavailable through the RLF GB (restricted to administrative use)
- can be restricted at subcategories or lower level objects

Rather than provide a set of built-in actions and provide a mechanism for augmenting this list with user-defined actions, the RLF makes no assumptions about the set of actions that might be useful to the modeler who is supporting a given community of users. All action processing is established through an action sub-model created using LMDL. As a guide to the creation of models wit actions, some of the sample models included with the RLF release contain a sample action sub-model. This model is shown graphically on SLIDE 64.

Slide 64



RLF network specifications include the declaration and use of Actions in order for model state information to be processed in a manner consistent with the type of state information, and to afford the opportunity for the RLF to provide larger scale asset processing. As shown in the previous slide, actions are declared as AdaKNET categories within a LMDL specification, but the graphical browser does not display them as categories since they are not used to model the domain, but rather to set up asset processing and manipulation within the domain model. This information may be confusing to an engineer using the graphical interface to interact with the library. SLIDE 65 identifies some library services that can be provided through action specifications.

Slide 65

Action Applications

- Execute tools
- View code
- Extract code
- Send mail
- Run prototypes
- Generate components

The library modeler should use one of the sample action sub-models as a starting point for defining the set of actions that will be made available to library users. As srefact-graph shows, there will be a top-level action category defined at the top of the library model (immediately below the root category). The RLF GB interface automatically suppresses the sub-network below the **Action Definition** category.

SLIDE 66 shows a LMDL description for the top-level of the action sub-model whose graph is shown in SLIDE 64.

Slide 66

Top Level Action Spec

```
category "Action Definition" (thing) is
end category;

category "Action Type" ("Action Definition") is
end category;

  category "System String" ("Action Type") is
  end category;

  category "Ada Procedure" ("Action Type") is
  end category;

  category "Message Pass" ("Action Type") is
  end category;

category Action ("Action Definition") is
  relationships
    has_action_type (1 .. 1) of "Action Type";
  end relationships;
end category;

category View (Action) is
  restricted relationships
    has_action_type of "System String";
  end restricted;
  attributes
    string is "xterm -e $RLF_PAGER ## &";
  end attributes;
end category;
```

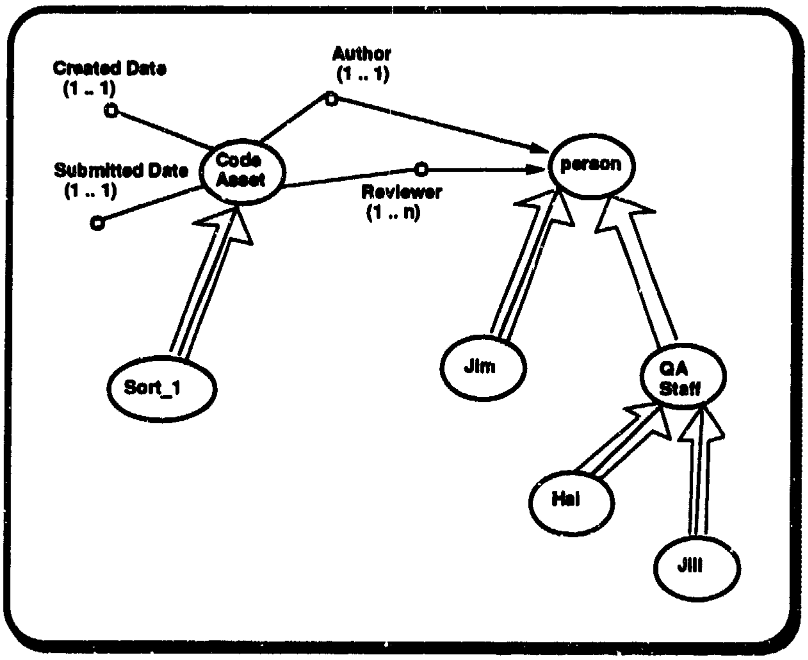
This specification shows the definition of a **View** action that enables the display the contents of a category or object which has textual state (e.g., the contents of a code file asset). Such a subcategory definition is required for each of the actions that the modeler wishes to include.

For an action provided through an OS-level command – identified as an action type of **System String** – the command to pass to the OS must be defined as a string attribute along with place holders for the arguments to be passed to the system command when the action is invoked. The string for the **View** action is also shown in SLIDE 66. In this example, the place holder is marked by the characters **##**. The character **&** is part of the system command indicating that the system command is to run as a background process.

Once a collection of actions has been defined in a sub-model as is shown above, their use within the library model is specified via a collection of related attribute and action definitions. A small model fragment will be used to help explain this connection.

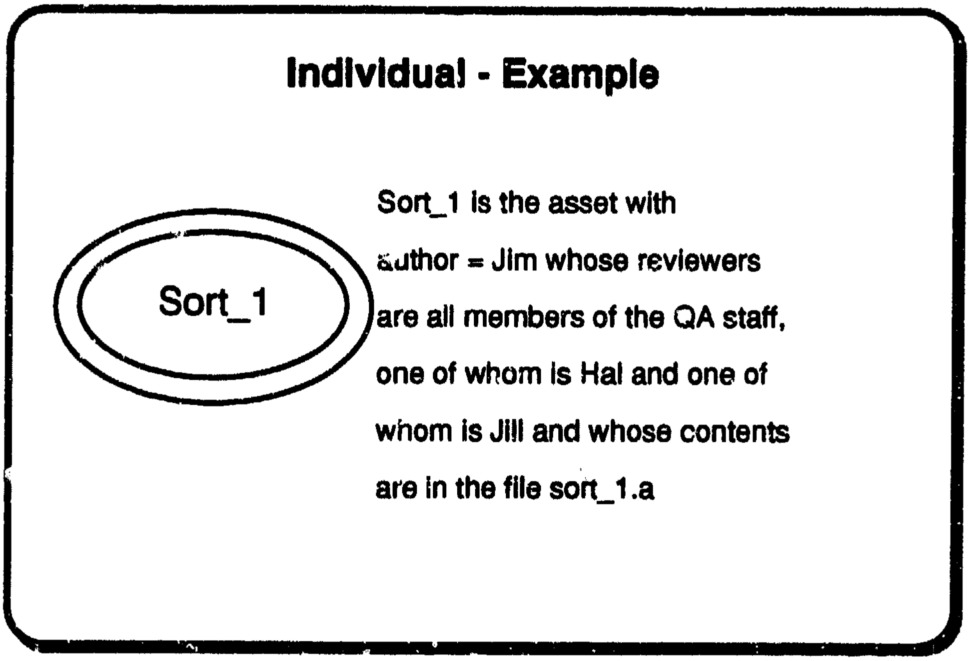
The slide sequence in the rest of this section illustrates the process of attaching asset contents to an existing model. A model fragment is shown that describes the fact that code assets have authors and reviewers and we wish to capture the text of an actual code asset within a library using this model.

Slide 67



Sort_1 is an individual asset which carries with it some internal state, including the source code which implements the asset.

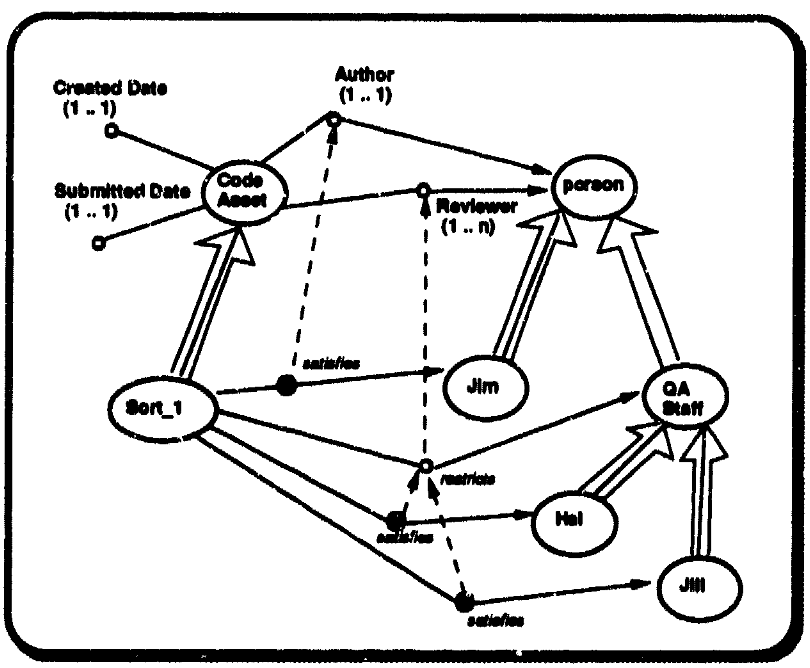
Slide 68



LMDL includes the ability to attach such internal state to an AdaKNET entity (category or object) and RLF actions (set up via LMDL declarations as shown above) are made available to the user to process this state. The file containing the source code for an asset is identified via an attribute of the asset category.

The various relationships that an asset is involved in are represented via filled relationships that connect the asset to other objects in the model. For example, the author for Sort_1 is Jim. This filled relationship is shown on the next slide. There are two reviewers identified for this asset – this information is also captured by filled relationships as shown in the slide.

Slide 69



SLIDE 70 gives a LMDL fragment which shows some of the LMDL clauses required to declare Sort_1 and to provide file contents for this asset.

Sort_1 LMDL Specification

```
object Sort_1 ( "Code Asset" ) is
  restricted relationships
    Reviewer (1..2) of "QA Staff";
    ...
  end restricted;
  fillers
    Jim satisfies Author;
    Hal satisfies Reviewer;
    Jill satisfies Reviewer;
    ...
  end fillers;
  attributes
    file contents is "sort_1.a";
    ...
  end attributes;
  actions
    "View Source" is View on contents;
    ...
  end actions;
end object;
```

Slide 70

SLIDE 70 makes the connection between the contents of the asset object (stored in the file `sort_1.a`) and the action to view the contents. This action is given the name **View Source** and is implemented via the system string defined as an attribute to the **View** action. The actual input for this action is provided by the **contents** attribute of **Sort_1**. The file name defined by this attribute is substituted at the location identified by the action placeholder when the action is selected by a library user who wishes to examine the contents of the **Sort_1** object.

Note that the action declaration shown on SLIDE 70 actually binds together the action with the state which the action is to process. The state is named in a LMDL attribute declaration which in many cases identifies a file that contains the state. This file must be supplied as an input parameter to the text of the action call.

This two-way connection is highlighted on the next slide. Note again that the slot in the action string where textual input is required is indicated by '##'. When the action is executed, the corresponding string value from the object provides the substitution for this placeholder.

Slide 71

Making Action Conections

Action at the visible object **Sort_1**:

```
"View Source" is View on contents;
```

Attribute at the invisible action **View**:

```
string is "xterm -e $RLF_PAGER ## &";
```

Selecting the **View Source** action brings up a viewer on the contents defined for the object. ## is a placeholder for the file name location of the contents.

The next slide notes the inheritance properties of actions and points out that the attribute to be processed by the action must be separately declared for each category or object for which the action can be meaningfully applied. Thus while actions are inherited, the targets for actions are not inherited.

Slide 72

RLF Action Inheritance

- Like relationships, actions are inherited down the specialization hierarchy
- Actions should be declared at most general category for which action is meaningful
- Attributes which are action targets are not inherited
- Action targets must be declared locally to each category or object for which action is meaningful

As an example of how to effectively use action inheritance, the view action defined above for **Sort_1** can be declared instead at the **Code Asset** category. All objects which belong to this category will have the view action defined for them. The contents attribute must be declared separately for each object which has contents. The RLG GB will only offer the action at those subcategories and objects which actually define the attribute. SLIDE 73 shows a partial specification with an action defined without a locally defined target.

Slide 73

Code Asset LMDL Specification

```
category "Code Asset" ( Asset ) is
  relationships
    Author (1..1) of Person;
    Reviewer (1..inf) of Person;
    ...
  end relationships;
  restricted relationships
    "Submitted Date" (1..1) of ... ;
    "Created Date" (1..1) of ... ;
    ...
  end restricted;
  attributes
    ...
  end attributes;
  actions
    "View Source" is View on contents;
    ...
  end actions;
end category;
```

The final step to creating models which use actions is to locate the attributes which provide inputs for the actions (files) in a place where the browser can find them. The RLF environment variable RLF_LIBRARIES is used to declare the base path name, and the state files must be located in a fixed location relative to this path as shown in SLIDE 74. The files can be physically be copied to this location or installed via the use of symbolic links.

Slide 74

Locating File Attributes

- RLF_LIBRARIES defines a relative location for all file state
- Files are located in the $RLF_LIBRARIES/Text subdirectory
- Files can be copied or linked to this location
- If multiple libraries are in use, a named subdirectory of $RLF_LIBRARIES/Text can be used
- Attribute declaration in LMDL must name the subdirectory
- E.G. file source is "sort_and_search/shaker_sort_.a";

8 Modeling Suggestions

The next few slides address some basic modeling issues and suggested practices that prospective RLF modelers should be aware of. These are summarized on the following slide.

Slide 75

Modeling Issues

- Model Evolution
- Naming categories, objects and relationships
- Representation choices
- Model Usage Goals

A suggested approach to take in the creation of RLF models is to start small and gradually evolve the model by adding detail. Since the models are meant to support users, it is useful to get intermediate versions of models in the hands of users and to solicit feedback from them. As will be shown in the next few sections of this tutorial, it is relatively easy to define a model and use the RLF GB to see it in action.

One important aspect of model evolution is the creation of intermediate specializations. The next slide outlines some useful cases to consider.

Slide 76

Intermediate Specialization

- Question linear model fragments (e.g. Z specializes Y, and Y specializes X)
  - what is rationale for intermediate category Y?
  - are there objects in category Y that are not in Z?
  - will model later include other categories of Y?
  - will objects in Z also belong to other categories unrelated to X
- Provide rationale for specialization hierarchy choices
- Choose carefully whether information should be modeled in specialization or aggregation hierarchy
- Reconsider categories which contain only single objects

Naming model elements is somewhat a matter of personal taste. Nonetheless, the next slide provides some suggestions that have proved useful in prior modeling activities.

Slide 77

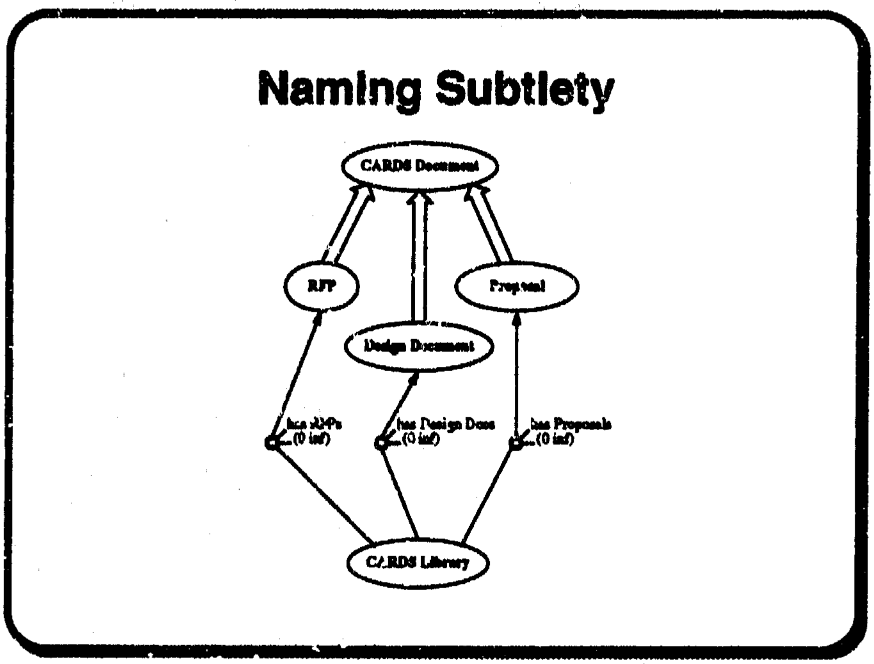
Naming Suggestions

- Don't get hung up on choosing perfect names -- model iteration and refinement of names should be expected
- Choose singular nouns as names of categories
- Choose active verbs or action noun phrases as names of relationships
- Avoid name repetition between relationship name and corresponding relationship type

  sometimes duplication is hard to avoid
- Accept that choosing names can be surprisingly challenging

The next slide illustrates a common naming dilemma.

Slide 78



Making representation choices can be especially troublesome for beginning modelers. There are often many factors to consider, and the modeler can have difficulty in deciding which RLF model building component to use. The next few slides illustrate some of these choices and view points to be taken.

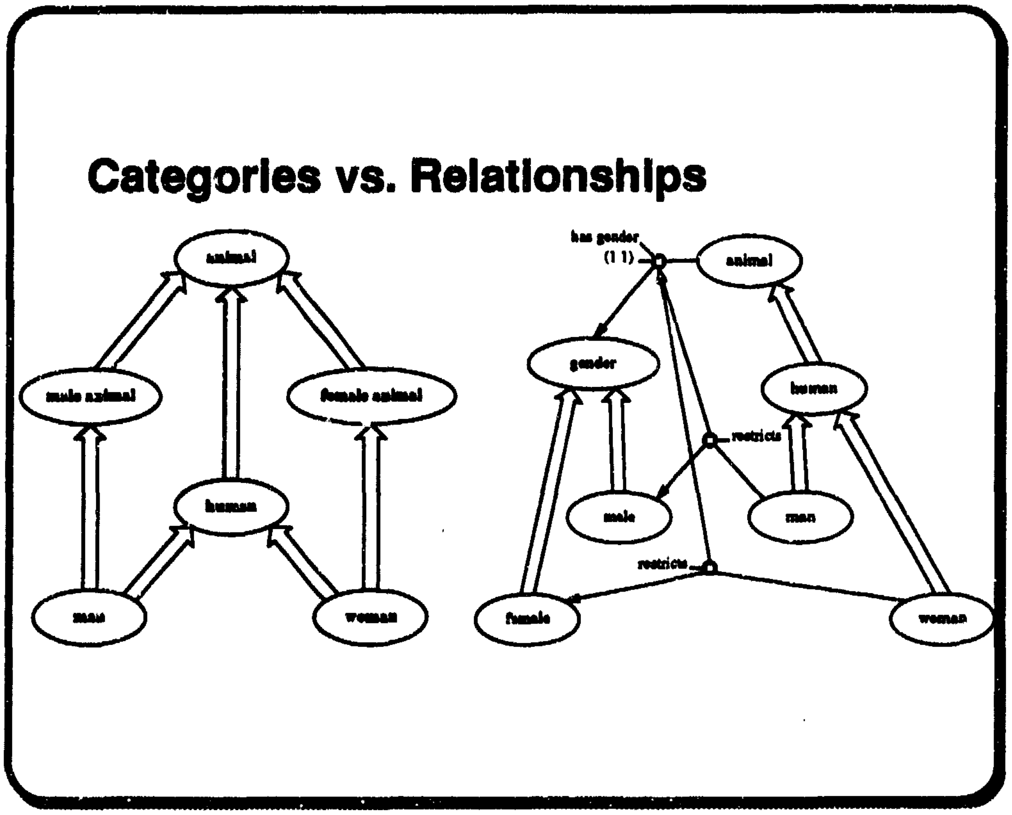
Slide 79

Representation Choices

- Categories vs. Relationships
- Relationships vs. Attributes
- Problem space vs. Solution Space
- Functionality vs. Implementation

The next slide shows out some candidate model fragments that represent alternate ways of capturing similar information.

Slide 30



In the model fragment on the left, the notion of gender is bound up in the distinction that animals exist as males and females. In the right model fragment, gender is extracted and made a model concept in its own right. Other categories can then be given restricted gender relationships to isolate male and female properties for those categories.

It is not always obvious how to model information that reflects category properties – both relationships and attributes can be used for this purpose. The next slide identifies some distinguishing features that may help in making modeling decisions. Relationships and attributes provide complementary capabilities for the modeler.

Slide 81

Relationships vs. Attributes

- Relationships are inherited; attributes are not inherited
- Relationships can be restricted and differentiated
- Relationships are processed only by built-in AdaKNET operations
- Attributes are unstructured and isolated to specific categories and objects
- Attribute information can be processed by external and internal actions
- Attributes are appropriate for long, descriptive properties

Another important distinction that the modeler should be aware of is that models ultimately must support the needs of a community of users whose viewpoints differ considerably from individual to individual, and from subgroup to subgroup. To illustrate this distinction, the next slide considers model focusing from the dual perspectives of problem space vs. solution space and functionality (features) vs. implementation. Models should always be developed with a clear understanding of the intended usage goals of the model, and the kind of application to be built on top of the model.

Slide 82

Model Focus Considerations

- Problem Space vs. Solution Space
- "What is to be accomplished" vs. "How to accomplish it"
- "Describing functionality and features" vs. "Describing implementations"
- There can be many different levels or spheres of functionality
- There can be many different implementation alternatives
- Should model lead to a single solution (e.g. automatic system composition) or support multiple solutions (e.g. human selection of system components)?
- Does model emphasize structure and features through specialization hierarchy or through aggregation hierarchy?

9 RLF Model Creation

The rest of this orientation package presents auxiliary material to accompany hands-on demonstration of the use of the RLF software to actually create and browse models. Except for some necessary context-setting for the declaration of inferencers to augment a network domain model, the discussion which follows is quite brief as it is written to provide annotation for the demonstration.

These notes cannot convey the full process of conducting a domain analysis whereby domain information is collected and evaluated for the purpose of identifying an organizational framework upon which a library of reusable assets can be figured. For the purpose of this orientation, it is assumed that a library of sorting algorithms and sort implementations is to be created and that a preliminary investigation of this application area (domain) has been conducted. Standard data structures and algorithms texts provide a written description of useful information to help decide which of several sort and search strategies and implementations can be used. The model built here is an attempt to capture a subset of information drawn from such sources and provide the means to help sort routine (re)users decide which of several approaches to take.

The notes attempt to convey the process of rendering the evolving model using the RLF. In particular, the model is presented in the stages indicated on the following two slides.

Slide 83

RLF Model Stages

- Build preliminary taxonomy -- show AdaKNET categories and specialization hierarchy
- Evolve taxonomy
- Add properties and essential relationships to taxonomy -- show AdaKNET relationships and aggregation hierarchy
- Attach objects to appropriate locations within taxonomy -- show AdaKNET objects and individuation links
- Provide object contents and object viewing capability -- show RLF state and action mechanisms

Slide 84

RLF Model Stages (cont)

- Review the model and expand/arrange as necessary -- complete network portion of model included in this tutorial release
- Provide model usage guidance and user support -- show AdaTAU inference base mechanism
- Complete full model definition

The RLF user must first set up an initial working environment. The steps necessary to do this are discussed in the RLF VDD. In particular, the user must have access to the basic RLF executables: RLF_GB, Lmdl and Rbdl and have established a working RLF data location identified by the RLF_LIBARIES environment variable. Object contents are stored in the Text subdirectory of the location identified by this environment variable.

Some of the necessary steps to work with RLF models are given on the next three slides.

Slide 85

Using Lmdl

- Set up your working directory

  via UNIX environment variable RLF_LIBRARIES

- $RLF_LIBRARIES sets *your* library directory
  - needs subdirectories /Text and /Taustuff
  - often referred to as the **Instances** directory
  - contains the Ada data structures that make up an RLF model and library

Slide 86

Using Lmdl (cont)

Sample working directory:

```
~nancy/Models
             /Instances
                       /Text
                       /Taustuff
             /model1
                    /model1.lmdl
                    /rbdl_stuff
                    /Text
```

Nancy needs access to executables Lmdl, Rbdl, Run_GB, Graphical_Browser, and necessary X resource definitions

Slide 87

Using Lmdl (cont)

- Create Lmdl specification using your usual text editor

  e.g. model1.lmdl

- execute `setenv $RLF_LIBRARIES` *`your-dir`*`/Instances`
- execute `Lmdl model1.lmdl`
- execute `Run_GB` to see results of model definition

For many of the model development stages shown in these notes, a network diagram is included that indicates some essential features of the model at this stage of its development along with a slide that shows some LMDL statements that define what is shown on the diagram. This package also includes a slide or two which summarizes the goals of each stage and points out interesting RLF features that are applied to meet these goals.

The first stage model builds a preliminary taxonomy of sorting algorithms.

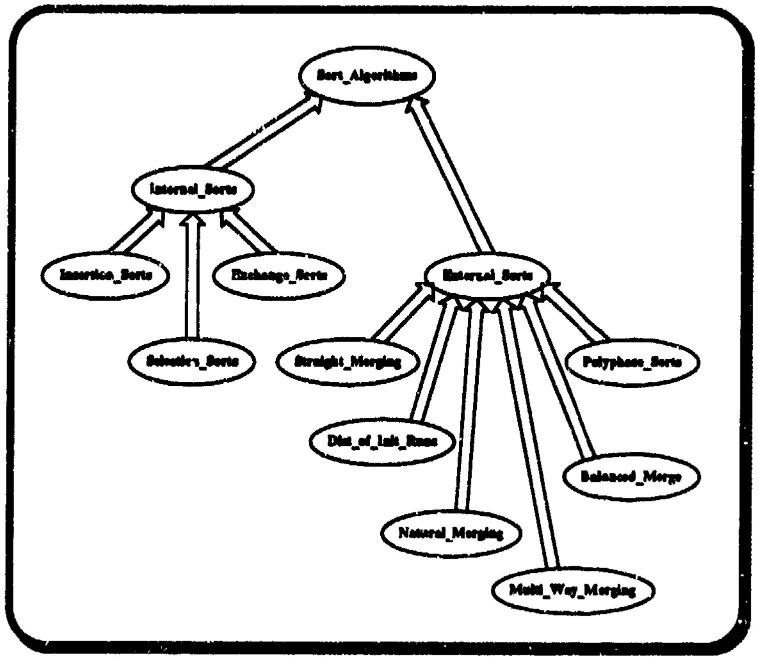
Slide 88

RLF Model Stage 1

- There is no one *right* way to begin
- One approach is to look for natural categorization within the domain
- For sorting, a natural distinction is between internal (e.g. array) and external (e.g. file) sorting
- Within these two categories, a number of subcategories are known (e.g Wirth's book: *Algorithms & Data Structures*)

One top-level decomposition is shown on the next slide and the complete LMDL specification file is shown in SLIDE 90.

Slide 89



Slide 90

```
library model stage_1 is

root category ( "Sort Algorithms" ) is
end root category;

category "Internal Sorts" ( "Sort Algorithms" ) is
end category;

category "External Sorts" ( "Sort Algorithms" ) is
end category;

category "Insertion Sorts" ( "Internal Sorts" ) is
end category;

category "Selection Sorts" ( "Internal Sorts" ) is
end category;

category "Exchange Sorts" ( "Internal Sorts" ) is
end category;
```

Slide 91

```
category "Straight Merging" ( "External Sorts" ) is
end category;

category "Dist. of Init. Runs" ( "External Sorts" ) is
end category;

category "Natural Merging" ( "External Sorts" ) is
end category;

category "Multi-way Merging" ( "External Sorts" ) is
end category;

category "Balanced Merge" ( "External Sorts" ) is
end category;

category "Polyphase Sorts" ( "External Sorts" ) is
end category;

end stage_1;
```

The next stage involves further decomposition of the preliminary taxonomy.

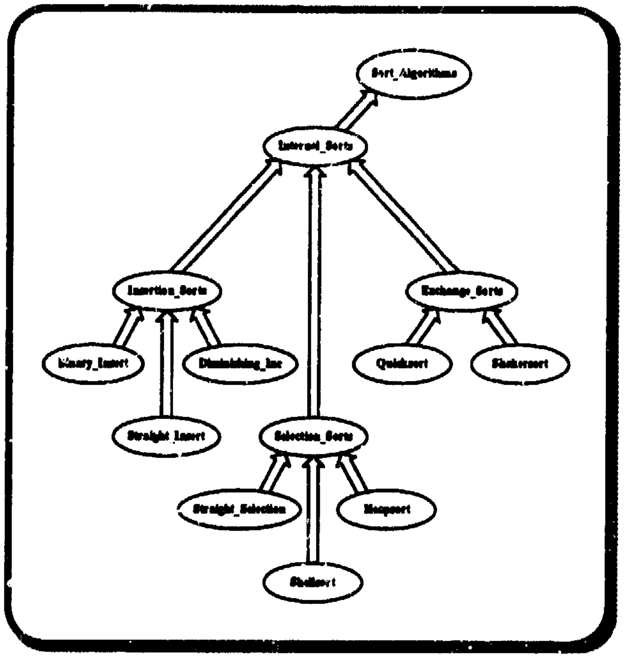
Slide 92

RLF Model Stage 2

- The top-level decomposition is too broad to be very useful
- The modeler must look to break down categories into subcategories which are closer in granularity to the categories of assets to be stored within the library
- Categories which correspond to known sorting algorithms are likely to be useful
- The next slide focuses on decomposing the internal sorts only

The next two slides slide show the decomposition of the internal_sorts three main subcategories into algorithm-level realizations of those three main categories.

Slide 93



Slide 94

```
library model stage_2 is

root category ( "Sort Algorithms" ) is
end root category;
. . .
category "Internal Sorts" ( "Sort Algorithms" ) is
end category;

category "Insertion Sorts" ( "Internal Sorts" ) is
end category;

category "Selection Sorts" ( "Internal Sorts" ) is
end category;

category "Straight Selection" ( "Selection Sorts" ) is
end category;

category "Exchange Sorts" ( "Internal Sorts" ) is
end category;

category "Binary Insert" ( "Insertion Sorts" ) is
end category;

category "Straight Insert" ( "Insertion Sorts" ) is
end category;
```

Slide 95

```
category "Diminishing Increment" ( "Insertion Sorts" ) is
end category;

category Shellsort ( "Selection Sorts" ) is
end category;

category Heapsort ( "Selection Sorts" ) is
end category;

category Shakersort ( "Exchange Sorts" ) is
end category;

category Quicksort ( "Exchange Sorts" ) is
end category;

end stage_2;
```

While the method shown in this section begins with an initial definition and elaboration of a model specialization hierarchy, some models may be more naturally built by concentrating on establishing a basic aggregation hierarchy for a small collection of categories and letting the specialization hierarchy grow as required. This choice is also somewhat based on the natural inclinations of the model builders themselves.

10 RLF Model Evolution

After producing a class hierarchy of the proper granularity, RLF models can then be configured to distinguish members of the taxonomy in terms of various class properties.

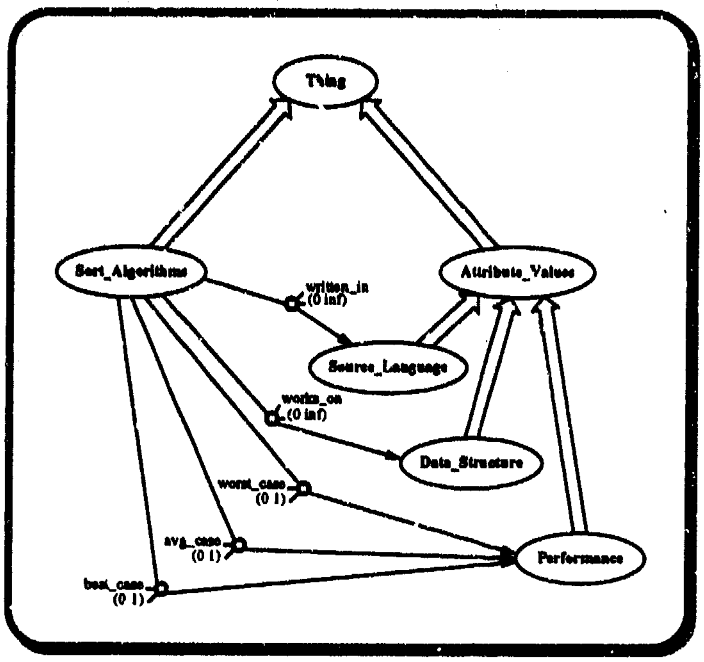
Slide 96

RLF Model Stage 3

- The RLF captures properties in terms of its aggregation hierarchy (relationships)
- Relationships have a name, a target type and a cardinality
- Relationships are defined for categories
- Subcategories inherit the roles of their parents
- Subcategories can restrict the target type and cardinality of inherited roles
- Objects which instantiate categories also need other objects to "fill" the relationships which the object has by virtue of its category membership

Once the definition of AdaKNET relationships begins, the use of the sort_algorithms class at the root of the specialization hierarchy no longer seems appropriate. In fact, many AdaKNET models begin with a generic *thing* root category so that somewhat unrelated sub-networks of the model can co-exist. In this example, a subtree is needed to locate and reference the value space for the various kinds of attributes used to classify sorting algorithms. This bifurcation is shown in SLIDE 97. SLIDE 97 also shows a collection of 5 relationships which are declared for the category sort_algorithms. The following slide gives the LMDL statements needed to establish the AdaKNET structure shown in the picture.

Slide 97



Slide 98

```
library model stage_3 is

root category Thing is          --| Define the root node
end root category;

category "Sort Algorithms" ( Thing ) is
  relationships
     is_written_in (0 .. infinity) of "Source Language";
     works_on (0 .. infinity) of "Data Structure";
     has_best_case_of (0 .. 1) of Performance;
     has_avg_case_of (0 .. 1) of Performance;
     has_worst_case_of (0 .. 1) of Performance;
     has_size_of (0 .. 1) of "Lines of Code";
  end relationships;
end category;

category "Attribute Values" ( Thing ) is
end category;

category "Source Language" ( "Attribute Values" ) is
end category;

category "Data Structure" ( "Attribute Values" ) is
end category;

category Performance ( "Attribute Values" ) is
end category;
. . .
end stage_3;
```

After a model's specialization and aggregation hierarchies have been fleshed out sufficiently, the modeler is in a position to address the identification and location of library assets – object instances of the categories already developed in the model.

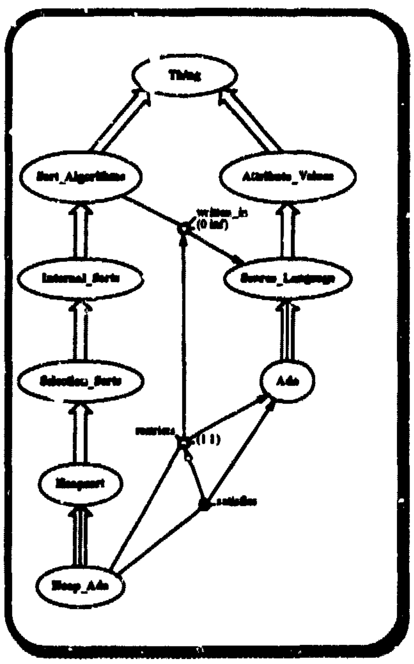
Slide 99

RLF Model Stage 4

- AdaKNET objects are declared as individuals in LMDL
- Individuals must declare their category parent(s) and any relationship fillers that are known
- AdaKNET permits partial descriptions for objects whose properties are only partially instantiated
- Any parent category relationship whose legal cardinality includes 0 may be omitted from the object's list of fillers.

A simple example of the declaration of an actual sort algorithm implementation and the filling of a relationship of the parent category by that object is shown in the next two slides. Note that the object **Heap Ada** restricts the cardinality of the *is_written_in* relationship it inherits from **Sort Algorithms** and then fills this relationship with an object belonging to the **Source Language** category.

Slide 100



The LMDL declarations defining the appropriate relationships and relationship restrictions are given in SLIDE 101.

Slide 101

```
library model stage_4 is
. . .
category "Sort Algorithms" ( Thing ) is
  relationships
    is_written_in (0 .. infinity) of "Source Language";
    works_on (0 .. infinity) of "Data Structure";
    has_best_case_of (0 .. 1) of Performance;
    has_avg_case_of (0 .. 1) of Performance;
    has_worst_case_of (0 .. 1) of Performance;
    has_size_of (0 .. 1) of "Lines of Code";
  end relationships;
end category;
. . .
category "Source Language" ( "Attribute Values" ) is
end category;

object Ada ( "Source Language" ) is
end object;

category "Internal Sorts" ( "Sort Algorithms" ) is
end category;

category "Selection Sorts" ( "Internal Sorts" ) is
end category;
```

Slide 102

```
category Heapsort ( "Selection Sorts" ) is
end category;

object "Heap Ada" ( Heapsort ) is
  restricted relationships
    is_written_in (1 .. 1) of "Source Language";
    works_on (1 .. 1) of "Data Structure";
    has_worst_case_of (1 .. 1) of Linearithmic;
    has_avg_case_of (1 .. 1) of Linearithmic;
    has_size_of (1 .. 1) of Number;
  end restricted;
  fillers
    Ada satisfies is_written_in;
    Array satisfies works_on;
    "N * log (N)" satisfies has_worst_case_of;
    "(N / 2) * LOG (N)" satisfies has_avg_case_of;
    Eighteen satisfies has_size_of;
  end fillers;
end object;
```

11 RLF Hybridization

Once model individuals have been identified, the modeler must address the contents of these individuals (the real assets being managed by the library), and the processing of asset contents appropriately. The RLF provides for the declaration of category and object state information and the declaration of actions to process the information.

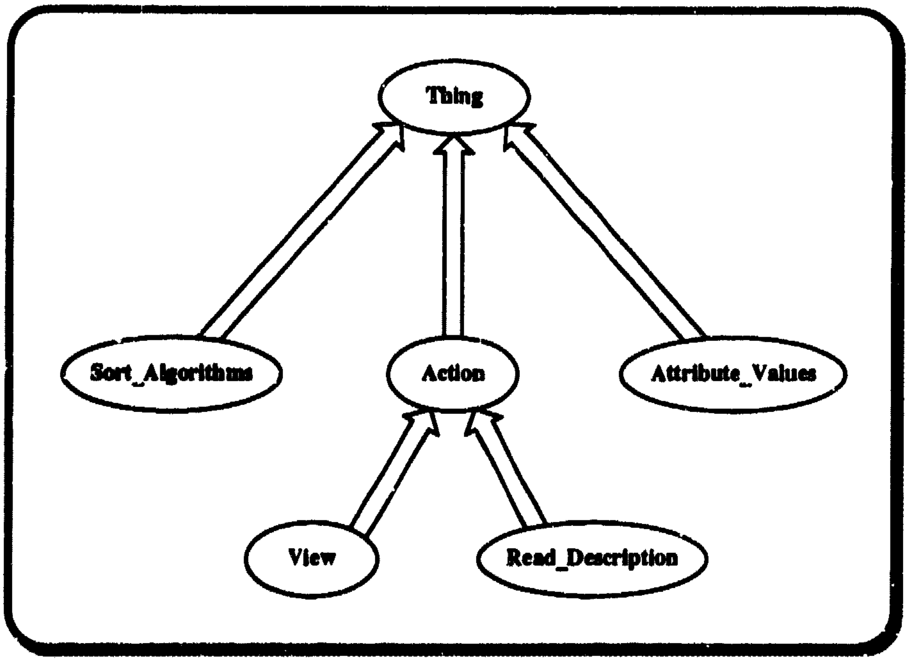
Slide 103

RLF Model Stage 5

- Both categories and objects can have contents (RLF state)
- State values can be integer or string valued, or can be the names of files which contain state information
- e.g., state attached to a subcategory of **Sort Algorithm** can explain some key features of the subcategory
- e.g., state attached to an asset object (AdaKNET individual) can point to a file which contains the code that implements the asset
- RLF actions are themselves declared within the overall domain model
- LMDL declarations connect pieces of state information to the actions that can process them
- Actions are inherited by subcategories, but state values are not

The model being demonstrated in conjunction with this tutorial only provides two simple actions to display source code of an asset and text files which describe an algorithm. The action subhierarchy of the model (derived from the root category) is shown in the next slide. SLIDE 105 declares these two actions and shows how they are connected to some examples of state information that they can process.

Slide 104



Slide 105

```
library model stage_5 is
. . .
category "Sort Algorithms" ( Thing ) is
  relationships
  . . .
  end relationships;
end category;

category "Action Definition" ( Thing ) is
end category;

category "Action Type" ( "Action Definition" ) is
end category;

  category "System String" ( "Action Type" ) is
  end category;

  category "Ada Procedure" ("Action Type") is
  end category;

category Action ( "Action Definition" ) is
 relationships
    has_action_type (1 .. 1) of "Action Type";
 end relationships;
end category;
```

Slide 106

```
  category View ( Action ) is
    restricted relationships
       has_action_type of "System String";
    end restricted;
    attributes
       string is "xterm -e $RLF_PAGER ## &";
    end attributes;
  end category;

    category "Display Description" ( View ) is
      attributes
         string is "xterm -e $RLF_PAGER ## &";
      end attributes;
    end category;

category "Internal Sorts" ( "Sort Algorithms" ) is
end category;

category "Insertion Sorts" ( "Internal Sorts" ) is
  attributes
     file desc_source is "sort_and_search/insertion_sort_desc";
  end attributes;
  actions
     "Read Description" is "Display Description" on desc_source;
  end actions;
end category;
```

Slide 107

```
object "Heap Ada" ( Heapsort ) is
  restricted relationships
  . . .
  end restricted;
  fillers
  . . .
  end fillers;
  attributes
     file desc_source is "sort_and_search/selection_sort_desc";
     file source is "sort_and_search/heap_spec_.a";
  end attributes;
  actions
     "View Source" is View on source;
     "Extract Source" is Extract;
  end actions;
end object;
. . .
```

A library model may have been initially created elsewhere but must be maintained and enhanced locally. The simple declarative syntax of LMDL makes it relatively easy to add new assets to a library change asset characteristics, install their contents and make viewing and processing the asset available to library end users. Currently, such changes require that the library's LMDL description file be edited and re-translated by the LMDL processor.

12 RLF Models and Submodels

Model creation and elaboration is not an exact science. Model revision is likely as model iteration proceeds. The stages shown in these notes do not provide examples of all RLF features, but they should give the beginning RLF model builder enough information to get started. The last stage considered here prior to considering the addition of RLF inferencing support to the model represents the decision to include searching algorithms as a natural extension to the sorting model.

The joining together of models is made easier by RLF v4.1 support for separate specification of model sections and the combination of these sections through incremental LMDL processing. Incremental LMDL is addressed in the Administrator tutorial. Thus, while the following example shows the embedding of the Search submodel within the original LMDL specification, it could be defined in its own specification file with the appropriate reference to the containing model.

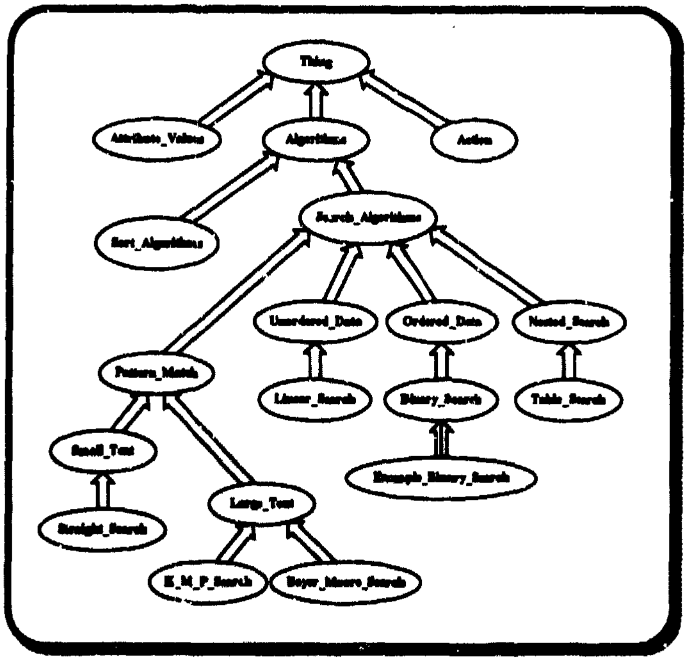
Slide 108

RLF Model Stage 6

- In practice, sorting and searching data structures go hand-in-hand
- The search taxonomy is analogous to the sort taxonomy
- Sort and search algorithms can have similar and related properties
- A user who requires a sort often requires a compatible search routine to go with it
- A new algorithms category is chosen as an intermediate class between the sort and search subnetworks

The complete model specification of the search and sort model is too large to be included here. A portion of the newly added search sub-model is shown on the next slide (only the specialization hierarchy is shown).

Slide 109



The complete **Sort and Search Algorithms** model is included among the sample models within the RLF v4.1 release.

1.3 RLF Model Inferencing

The next series of slides discusses the basic features and modeling approach provided by the AdaTAU inferencing subsystem. As mentioned earlier, AdaTAU provides the ability to "program" how a user can make effective use of the information within a complex network model and to explain key features of the model, thereby helping a user to learn about the model.

Slide 110

Fact Bases in AdaTAU

- Basic AdaTAU object is an "inference base" or "inferencer"
- Composed of "fact bases" and "rule bases"
- Fact bases are collections of "facts"
  - Facts are items represented as (attribute, value) pairs
  - Facts are restricted according to a defined fact base schema
  - Facts can have one-and-only-one, some, or arbitrary (a,v) pairs as assertions as dictated by the schema
  - An assertion is when a fact, given as (a,v) pair(s) are installed in a "fact base"

In addition to facts about the domain model and the user's perception of the model, AdaTAU provides rule bases to process fact bases and thereby expand the what the user knows about the model and its contents.

Slide 111

Rule Bases in AdaTAU

- Rule bases are collections of "rules"
- A rule is said to be "fired" when its action is performed because its antecedents are asserted
- There are three kinds of rules with different actions
  - Irules (inference rules) - assert "consequent" facts
  - Qrules (question rules) - queue a question to ask the user
  - Frules (focus rules) - queue suggestion to switch inference bases

The content of an AdaTAU Inference Base is determined by a set of declarations that define the needed AdaTAU fact and rule base structures. These declarations are written using RBDL – Rule Base Description Language – and consist of sections as shown on SLIDE 112. An individual RBDL file may in fact include only a few of these sections the majority of them are optional.

Slide 112

Inference Base Components

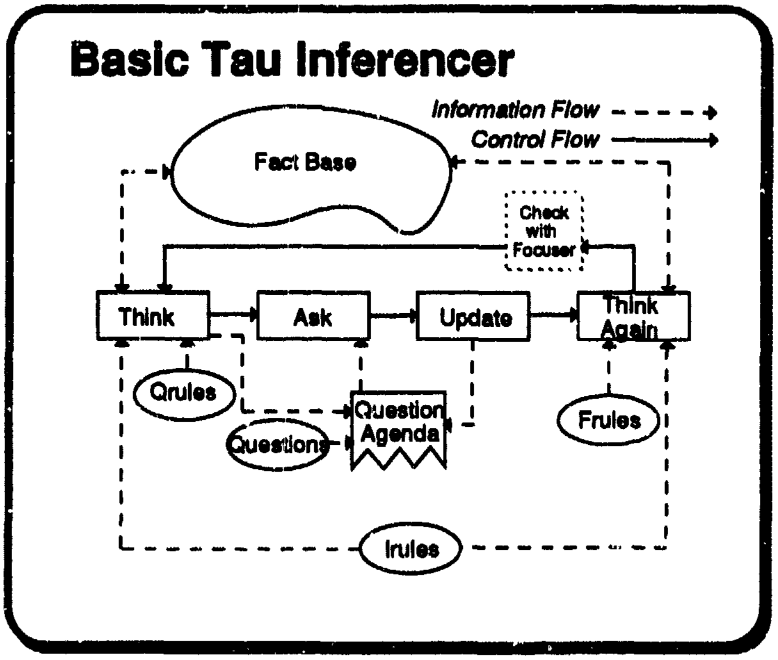
- fact base schema - describes facts and potential (a, v) pairs
- initial fact base - facts asserted at startup
- fact parameters - describe facts passed between between inferencers
- irule base - collection of irules
- qrule base - collection of qrules
- question base - collection of questions queued by qrules
- frule base - collection of frules

AdaTAU operations are done in a cycle until nothing more can be done:

- Think - fire any rules whose antecedents are asserted
- Ask - choose the highest priority question and ask it
- Update - add consequent facts of questions to local fact base

This inference process is summarized on the following slide. Note that the current AdaTAU inference cycle is just one of many possible cycles that could be implemented using the RLF software. Changing the inference cycle requires modifying RLF Ada code at a fairly high level so that RLF application builders can modify it themselves.

Slide 113



In distributed AdaTAU (DTAU) you also see if a switch between inferencers has been queued; if so the fact parameters are exported to the suggested inference base and in most circumstances the user continues the inference process in the new inference base context.

AdaTAU, like AdaKNET, has complex set of Ada interfaces and a specification language is provided to insure that the interfaces are used correctly.

Slide 114

Rule Base Description Language – RBDL

- Ada-like non-procedural language
- Specifies all seven AdaTAU elements
- Is translated to produce files which represent an inference base
- Is modified and re-translated to change the inference base

The last two stages of the **Sort and Search Algorithms** model evolution touch on the provision of more heuristic, procedural information aimed at guiding (advising) users about the domain model and the assets contained with the library. This sort of information is captured by using the RBDL language within the AdaTAU subsystem of the RLF. AdaTAU can be applied to guide users through a structurally complex network model and direct them to look for and extract information in the manner that a domain expert might use.

In actuality, the demonstration model used here is probably too simple to require the provision of such a guided mode of interaction but the next two stages will explore the development of such a mode to illustrate the application of AdaTAU.

Slide 115

RLF Model Stage 7

- A complex model may bewilder inexperienced users
- Extracting components from such a model can be tedious
- Solution is to provide supportive guidance in the navigation and use of the model
- AdaTAU inferencers are attached as state to AdaKNET concepts and/or individuals
- RLF provides built-in inferencing "action" to process these inferencers
- Inferencers provided auto-navigation

SLIDE 116 shows an LMDL fragment that attaches some inferencers to the demonstration model.

Slide 116

```
library model stage_7 is

root category Thing is          --| Define the root node
end root category;

category Algorithms ( Thing ) is
  relationships
     is_written_in (0 .. infinity) of "Source Language";
     . . .
  end relationships;
end category;

attach inferencer algorithms to Algorithms;

category "Sort Algorithms" ( Algorithms ) is
end category;

attach inferencer sort_algorithms to "Sort Algorithms";
. . .
```

The next few slides consider the sort_algorithms inferencer in some detail. During the demonstration, other initial inferencer versions may be shown to show AdaTAU and RBDL features in additional detail. In SLIDE 117, the basic structure of the facts that this inferencer can process are declared.

Slide 117

```
fact base schema schema_sort_algorithms is
  prior_context : one_of (algorithms, sort_algorithms,
        internal_sort, external_sort, system, unknown);
  answer : one_of (internal_sort, external_sort,
        dont_know, still_dont_know, unknown);
  continue_confirmed : some_of (yes, no);
  no_hits : one_of (yes, no);
end schema_sort_algorithms;

fact parameters is
  imports : (prior_context => mandatory);
  exports : (prior_context => focal);
end fact parameters;

initial fact base init_facts_algorithms is
  (answer, unknown),
  (continue_confirmed, no),
  (prior_context, system);
end init_facts_algorithms;
. . .
inferencer sort_algorithms is
  qrule base : qrules_sort_algorithms;
  frule base : sorting_frules;
end sort_algorithms;
```

SLIDE 118 declares some of the questions that the inferencer may ask users while they are interacting with the inferencer.

Slide 118

```
question base questions_sort_algorithms is
  question sort_type_selection is
    text : { Select type of sort algorithm, internal for sorting
             an array for example, external for date on tape.};
    type : one_of;
    responses :
       "Internal" => ~(answer, unknown), (answer, internal_sort);
       "External" => ~(answer, unknown), (answer, external_sort);
       "Don't know" => ~(answer, unknown), (answer, dont_know);
  end question;
. . .
  question give_up_question is
    text : { There is no further advice I can give without a
           selection at this point.};
    type : one_of;
    responses :
      "Confirm" => (no_hits, no);
  end question;
end questions_sort_algorithms;
```

Questions are posted to an agenda that is under the control of the AdaTAU inferencer engine as a result of the execution of QRules. The structure of Qrules is illustrated in SLIDE 119.

Multiple AdaTAU inferencers may be declared to lie within an RLF model. Focus rules (FRules) within individual inferencers direct the user's attention from place to place within the model, deducing appropriate facts as these transfers of focus take place. An example of such a rule is also shown in SLIDE 119.

Slide 119

```
qrule base qrules_sort_algorithms (questions_sort_algorithms) is
  qrule sort_type_selection is
    antecedent : (answer, unknown);
    question : sort_type_selection;
    weight : 1;
    justification : (Determine whether or not internal or
                     external sorting algorithms are desired.);
  end qrule;

  qrule clarify_question is
    antecedent : (answer, dont_know);
    question : clarify_question;
    weight : 1;
    justification : (To give an explanation and prompt for a
                     choice again.);
  end qrule;

  qrule give_up_question is
    antecedent : (answer, still_dont_know);
    question : give_up_question;
    weight : 1;
```

Slide 120

```
    justification : ( Cannot proceed without further input.);
  end qrule;
end qrules_sort_algorithms;

frule base Sorting_Frules is
  frule Internal_Interest is
    antecedent : (answer, internal_sort);
    export : (prior_context, sort_algorithms);
    focus : internal_sorts;
    weight : 1;
    justification : (Since advice on internal sorting algorithms
                     is desired, we will move there.);
  end frule;
end Sorting_Frules;
```

The structure of an inferencer is directed to ask the user appropriate questions about the user's intentions with respect to the library assets which are stored in terms of the available library model. The model builder structures questions and forms rules in each of the available rule classes to elicit sufficient information from the user so that the user's attention is drawn to assets that are likely to meet the users needs.

The last stage of the model building stages shown in this orientation package is the complete **Sort and Search Algorithms** model including one inclusive LMDL file and fourteen separate RBDL files. The complete text of these files is too large to include here. During the orientation, the operation of a library based on this model was demonstrated.

A good place to begin to explore the RLF is to take the basic specification files discussed here and elaborate them in various ways. In particular, additional inferencer specifications can be provided that direct users to access the model to meet other user requirements.